UNITED STATES DEPARTMENT OF THE INTERIOR

GEOLOGICAL SURVEY

SAKI: A Fortran program for generalized linear inversion of gravity and magnetic profiles

by

Michael Webring

Open File Report 85-122

1985

Table of Contents

Abstract

A gravity and magnetic inversion program is presented which models a structural cross-section as an ensemble of 2-d prisms formed by linking vertices into a network that may be deformed by a modified Marquardt algorithm. The nonuniqueness of this general model is constrained by the user in an interactive mode.

Introduction

This program fits in a least-squares sense the theoretical gravity and magnetic response of a geological structure model to a profile of observed data. The model consists of an ensemble of n-sided polygons that map in cross-sectional form the interpretated variations in subsurface density and magnetic properties. The third dimension of the model extends at a right angle to the cross section a distance sufficient to avoid edge effects. Model parameters which can be varied are the location of prism vertices and the density or magnetic property of each prism.

SAKI is an interactive gravity and magnetic profile modeling program that uses generalized linear inversion to improve model parameters. Both fields may be inverted simultaneously to gain added constaints on the model. The problem of nonunique solution must be handled by the user, who must not only have some knowledge of the geology but should have a feeling for the gravity and magnetic responses of simple bodies and for which parameters of the model are good candidates for improvement. The automatic inversion capability is a time saver in that a group of parameters may be changed in a systematic way and the response noted by the user. The improved parameters frequently diverge from the starting model and by studying what the inversion algorithm is trying to improve the user can pose a geologically feasible alternative. On the other hand, where the model is close to convergence, a slight change of many parameters can cause a dramatic improvement in agreement between calculated and observed curves.

The program has several additional features that ease the problem of interpretation. A function is available that produces synthetic data to which noise can be added so a user can familiarize himself with program control and factors that constitute a well-posed inversion. Second, groups of individual prism responses can be plotted that allow the user to see how two or more curves combine to produce various features of the overall curve. An internal plotting package using low-level scale, line, and character calls allows on-line plotting to video terminals or to off-line devices with the number of specified commands varying from completely automatic to user specified parameters entered either interactively or through a command file. A hardware dependent graphic input section using either cross-hair, light pen, digitizing pad, or mouse should not be difficult for an experienced programmer to add to the system.

General linear inverse theory

The discrete response of a potential function can be represented by

$$g_{c_i} = \sum_{j=1}^{n} x_j \, a(y_i, y_j) \qquad \begin{matrix} i = 1, \text{ m observations} \\ j = 1, \text{ n bodies} \end{matrix} \qquad (1)$$

where x_j is the linear parameter of density or magnetization and $a(y_i, y_j)$ is the nonlinear term descibing the geometric effect of body 'j' at observation 'i'

or in matrix notation $G = AX.$ (2)

if O represents the observed data vector then the error $E = O-G = O-AX$

or $$e^2 = E^+ E = [O-XA]^+ [O-XA]$$

$$e^2 = O^+ O - 2X^+ A^+ O + X^+ A^+ AX$$

and minimizing the squared error function

$$\frac{\partial e}{\partial x} = -2A^+O - 2A^+AX = \emptyset$$

$$A^+AX = A^+O$$

A^+A is a positive definite matrix which can be inverted to solve for the linear parameters x_i.

$$X = [A^+A]^{-1} A^+O \qquad (3)$$

This inversion is commonly ill-posed and generates large parameter excursions that may be avoided by adding a term sensitive to parameter change to the error function (Marquardt, 1963)

$$e^2 = E^+E + k^2X^+X$$

which yields $$X = [A^+A + k^2I]^{-1} \; A^+O. \qquad (4)$$

4 is the familiar damped least-squares solution for linear parameters. K is the system damping parameter and serves to stabilize the system at the cost of increasing the value of e^2.

Generalizing 4 to solve for the nonlinear body geometry parameters contained in $a(y_i,y_j)$ of (1).

$$g_{c_i} = A_i \; (p_j)$$

Where p_j is an n vector of linear or non-linear model parameters. Expanding in a Taylor series about some initial value p_j^o.

$$g_o \simeq g_c^{\,1} = g_c^o + \left.\frac{\partial A_i}{\partial P_j}\right|_{p_j^o} \Delta p_j + \ldots.$$

$$e_i = g_{o_i} - g_{c_i}^o = \left.\frac{\partial A_i}{\partial p_j}\right|_{p_j^o} \Delta p_j \qquad (5)$$

In matrix notation

$$E = 0 - G = A\,\Delta P$$

A is an m by n Jacobian matrix of partial derivatives with respect to the varying parameters. Substituting into equation 4 we have

$$\Delta P = [A^+A]^{-1}\ A^+E \qquad \text{for } m > n$$

and the damped form

$$\Delta P = [A^+A + k^2I]^{-1}\ A^+E \qquad (6)$$

Equation 6 is the standard form of Marquardt (1963).

The effect of the damping parameter k may be seen when the A matrix is rotated into a coordinate system where the n parameters are independent (Jackson 1972). This rotation decomposes A into a product of two orthogonal matrices (U and V) and a diagonal matrix of eigenvalues (Penrose, 1955, Lanczos,1961). The dimension of U is m by n, V is n by n, and S is n by n.

$$A = U\ S\ V^+.$$

$$\text{now}\quad A^+A + k^2I = VSU^+ \cdot USV + V(k^2I)V^+ = VS^2V^+ + V(k^2I)V^+ \qquad (7)$$

and substituting 7 into 6

yields

$$\Delta P = [VS^2V^+ + V(k^2I)V^+\ ]^{-1}\ VSU^+\ E$$

$$\Delta P = V\ [S^2 + k^2I]^{-1}\ SU^+\ E$$

$$\Delta P = V \frac{s_i}{k^2 + s_i^2}\ U^+\ E \qquad (8)$$

where s_i are the diagonal elements of S.

System damping can be seen to be smooth attenuation of eigenvalues centered on some threshold k. Jupp (1974) demonstrates that regarding the damping as

$$\frac{s}{k^2 + s^2} = \frac{s^2}{k^2 + s^2} \cdot \frac{1}{s} = \frac{s^{2n}}{k^{2n} + k^{2n}} \cdot \frac{1}{s} \qquad (9)$$

and varying n allows the damping rate to be tailored to a particular inversion. The Marquardt algorithm (eq. 8) has an 'n' of one, a slow attenuation rate which allows leakage of small eigenvalues (representing ill-defined or fine detail parameters) into the system. This program uses an 'n' of 2 but the algorithm is stable for small numbers and the exact value is not critical. Performance is erratic for numbers greater than about 4. Figure 1

shows three attenuation curves for n=2. The attenuation governing which eigenvalues are allowed into the correction vector becomes sharper in the range where unstable parameters cluster.

The theory outlined above may be implemented by either equation 6 or 8, both have pros and cons. The advantage of 6 is ease of computation with the penalty of recomputing the matrix inverse for each trial damping factor. Using 8 the Lanczos decomposition is calculated once and then damping factors applied using matrix multiplication. The decomposition requires more CPU time and a stable alogorithm (Golub, 1970), but returns the benefit of flexibility and extra information.

The damping is optimized in both methods because an underdamped system can invalidate the linearity assumption of equation 5 and an overdamped system wastes time by making small changes. Once a damping factor is chosen a forward calculation is performed to check the RMS error. Several damping factors can be tested and the shape of the error curve analyzed to determine the optimal system damping. Test forward calculations are a convenient way to check the optimization procedure, but not absolutely neccessary as long computation times should to be avoided if the program is interactive.

A second approach compares the direction of the trial update vector with the so-called steepest descent vector. The steepest descent vector is the result of an highly damped system and is characterized as always leading to convergence at the cost of a small step size. Experience has shown the angle between these two vectors reaches a minimum at a slightly more conservative damping than the mimimum RMS error. Figure 2 depicts the common case where choosing the minimum RMS error would result in a larger correction vector. SAKI uses both the RMS error and the angle in a binary search pattern to select a damping factor.

Parameter excursions are limited to set fractions of the overall model length and depth to keep the model in the same 'neighborhood'. Damping factors are also increased if a vertex enters another body.

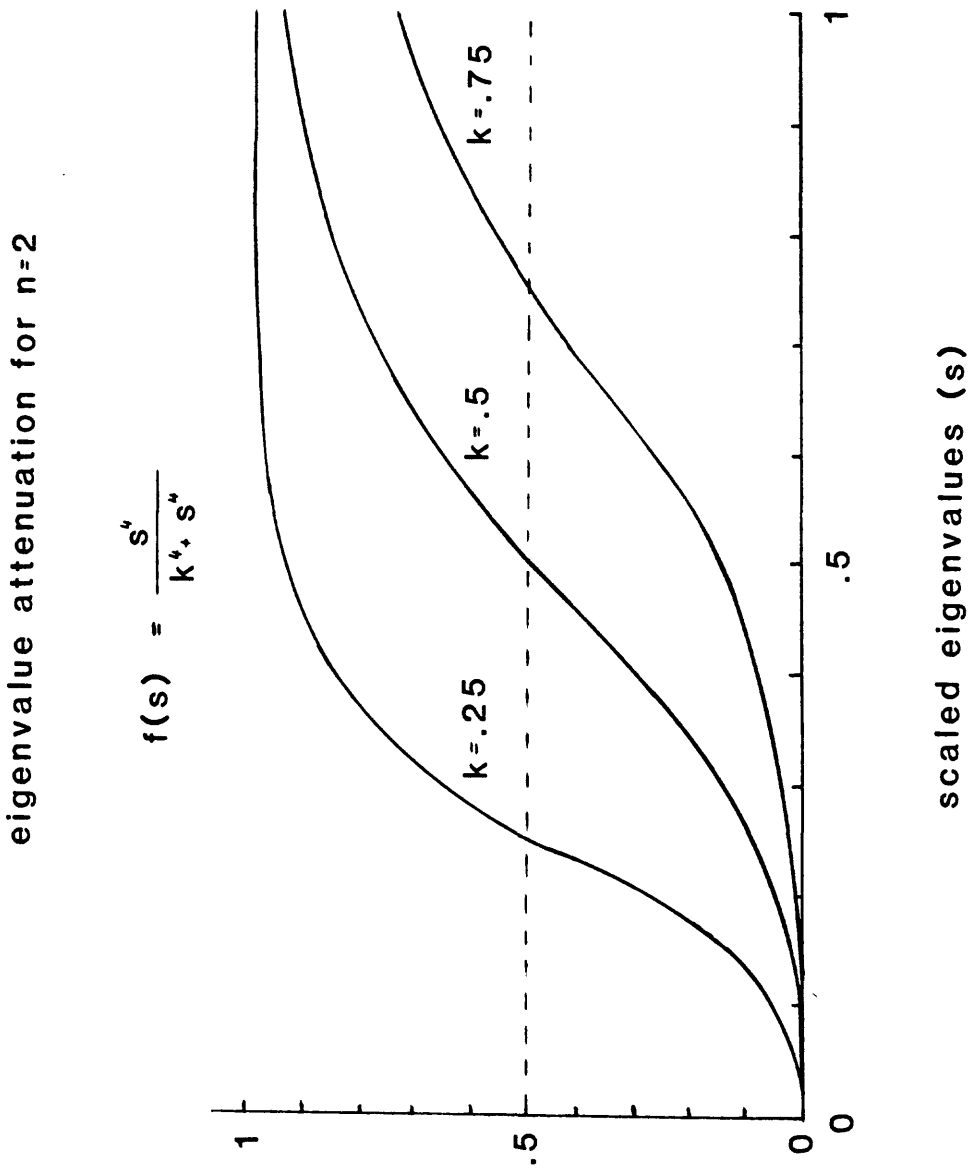

eigenvalue attenuation for n=2
f(s) = s^4 / (k^4 + s^4)
k=.25
k=.5
k=.75
1
.5
0
0
.5
1
scaled eigenvalues (s)

figure 1

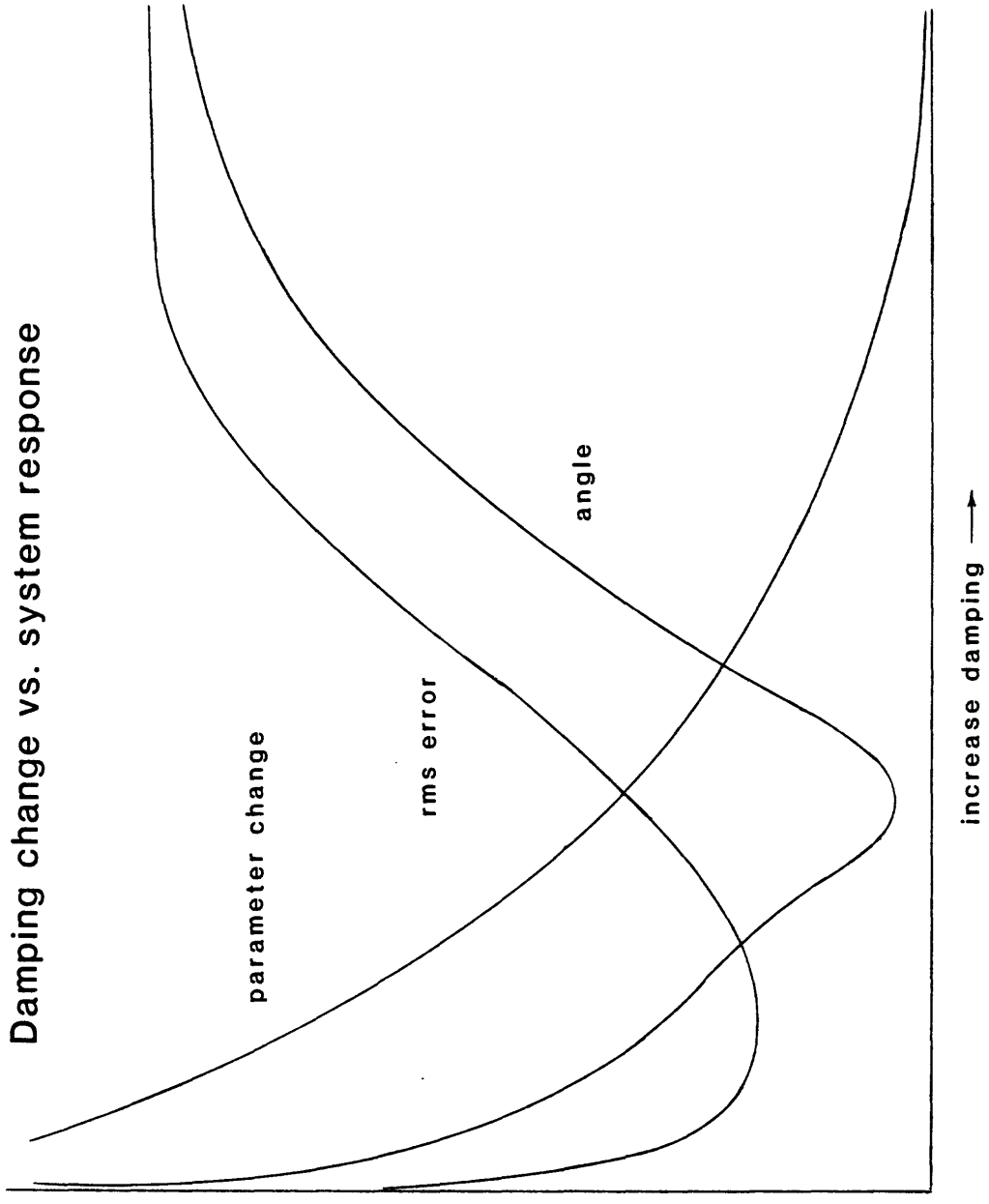

Damping change vs. system response
parameter change
rms error
angle
increase damping

figure 2

USER'S GUIDE

There are three input files. The first input file describes the model as a cross-section through a 2-d prism ensemble. The second is an optional observed gravity and/or magnetic data files. The third is an optional command file that contains control parameters.

The coordinate system is referenced to the profile. X is horizontal distance in kilometers along the profile and for magnetics is south to north unless modified by the 'azmuth' parameter. Z is depth, positive down in kilometers; both data and model share the same arbitrary origin. The Y coordinate is used to describe the strike length of the prisms and is oriented positive toward the viewer when X increases to the right.

Density contrasts should be referenced to the Bouguer gravity reduction density. For instance when modeling a body with a real density of 2.5 and the Bouguer reduction density is 2.67, use a model density of -.17. A Bouguer density close to the overall surface density of your model is the best choice because this allows bodies close to the station locations to have contrasts nearly equal to zero, minimizing the error caused by simulating the terrain with a 2-d model.

Models should extend beyond the ends of the profile 5 to 10 times the depth of the deepest body to minimize edge effects. The strike lengths of the bodies should follow the same guideline. A near surface body, for instance an alluvial deposit, could have a strike length that matches the mapped extent; while the basement formations would be longer depending on depth. Gravity calculations may be done for asymmetric bodies (ie. not centered on the profile), but experience indicates that contraints are usually not available to make the added computation worthwhile. Asymmetric calculations are not available for magnetic bodies.

Model File

limitations: 50 bodies, 100 vertices

The model file is an ASCII disk file that is initially generated with a standard text editer by the user prior to running the program. The program reads the model file with free field formatting, meaning the numbers may be located anywhere on a line. Several of the lists are terminated by non-alphanumeric characters, freeing the user from specifying the length of the lists.

The model is defined by a set of vertices that are linked together by lookup lists to form a network. No line segment of the network may cross another, and no vertex may be inside a polygon that defines a body. The first section of the model file is the vertex list whose entries are the x and z coordinates in kilometers for each vertex. The list is terminated with a non-alphanumeric character and may have up to 100 coordinate pairs.

The prism definitions follow the vertex list and consist of two parts.

part 1 This line must have all 7 numbers present.

density in gm/cc
susceptibility in cgs. Refer to the magnetic parameter section of the command file to define the direction of the induced magnetic vector.
prism lengths in km (right cartesian coordinates ie 20, -20), The first number is the near side and the second the far side. These lengths may be different and will move the prism off axis for gravity calculations. Zeroes will cause a default to an infinite 2-d calculation. The default value is the same as parameter 'plenth', which is explained in the general parameter section.
remanent magnetization in emu/cc
remanent inclination in degrees downward from horizontal.
remanent declination in degrees clockwise from North.

The total magnetization used in the forward calculation is a vector sum of the induced and remanent magnetization or Ji + Jr, where Ji = susc * earth field strength in nTesla and Jr = remanent * 100,000.

part 2 is a lookup list defining the sequence of vertices that make up a prism. Use clockwise order, do not repeat the first index, terminate the list with a non-alphanumeric character. After the terminater is an optional 36 character body identifier, if it's blank the program will number the bodies. Several lines may be used in the lookup list.

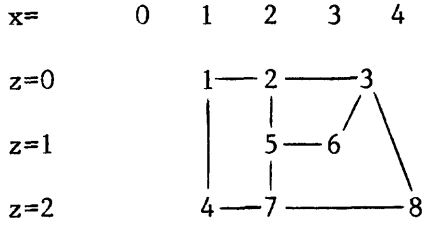
This data structure was developed for the inversion in X and Z of the vertices or nodes of the model network. The coordinates of each node are stored in one location, and when those coordinates are changed the bodies associated which that node are implicitly updated. A model is checked for validity before a forward calculation is performed, and error messages are printed to warn the user should something be amiss. Model errors are not fatal and the program will allow plotting and editting of node locations.

Note: errors in the body lookup list must be corrected outside the program with a text editor.

Example

There are two parts to the model file:
(1) the vertex array defining x,z locations for prism corners
(2) prism specifications, each with two sections - a physical property line and a lookup list specifying which vertices make up the body.

The example plot below has three prisms between x coordinates of 1 and 5, and z coordinates 0 and 2,



and would have a model file that looks like

```
1,0  2,0  3.5,0            [1,0 is vertex one; 2,0 is vertex two; etc]
1,2  2,1  3,1   2,2  4.5,2
>>body specifications
-.2 2.e-3  20 -20  1e-5 60 18 <<phys_prop
1 2 5 7 4 << left side body
.2  3.e-3  20 -20  1e-5 -120 18 <<phys_prop
2 3 6 5 << center top
0.1  0.0  0 0  0 0 0 <<phys_prop
5 6 3 8 7 << right side
```

Here there are 8 prism vertices arranged in x,z pairs and typed in free field with any number per line. The program will try to reject models that have void areas and overlapping prisms. A prism with a density of zero is legal to link sections of the model. Note that vertex 5 is present in all three prisms, do not skip a vertex merely because the prism side does not change direction. During modeling the network topology described by the linked vertices remains the same even though the prism shapes may change due to vertices being moved.

The '<<' are any non-alphanumeric character and used as delimiters, note that they are present at the end of the vertex and lookup lists. At the end of the physical property line they are optional. Blanks or commas are necessary to separate numbers from each other and from the delimiters, the program counts entries in each list.

Observed Data Files

The observed gravity and magnetic data are contained in two separate disk files; each line contains a horizontal distance, depth, and gravity or magnetic value. The combined number of observations is less than 400. The values may be in free field format (ie. anyplace on the line). A profile is assumed to be straight and therefore has a single azimuth to which the magnetic field components are referenced.

Command File

The command file is optional. Enter a carriage return and the program will prompt for the names of the input model, gravity, and magnetic files, the other parameters have defaults. The command file is a 'namelist' that many Fortran compiled programs recognize.

example where the namelist title is 'parms':

```
&parms
mfile='test.mod', gfile='test.grv', hfile='test.mag'
azmuth=45, efield=54495., edec=17., einc=65.
&
```

file control parameters

parameter	description
mfile	model filename, default is blank and program will prompt
gfile	gravity data file (optional).
hfile	magnetic data file (optional). If neither gfile or hfile is specified the program assumes you want a synthetic run and will prompt to obtain observation locations.
ifmtg	input format for gfile, specify 3 floating fields for x,z,and gravity. Example ifmtg="(2f10.3,27x,f7.2)", default is free field.
ifmth	input format for hfile, specify 3 floating fields for x,z,and magnetic values. Example ifmtm="(3f10.3)", default is free field.

magnetic parameters

compon	the calculated magnetic component. Set to x, y, z, or t where t is total field. Example: compon='t' (default).
efield	Earth's field strength (default = 50000 nTesla).
einc	Earth's field inclination in degrees, positive down from horizontal (default= 90).
edec	Earth's field declination in degrees clockwise from north. Example for north 45 degrees west: edec=-45 or edec=315. (default= 0).
azmuth	profile orientation, degrees clockwise from north (default=0 or a profile that goes from south to north).

The remanent magnetization declinations (from the model file), the Earth's field declination, and the profile azimuth are the only parameters that are tied to geographical coordinates.

general parameters

iave,jave (for gravity and magnetic respectively) automatic DC average removal from the calculated field where the average is the difference between the calculated and the observed functions. Set not equal to zero to activate, each time the forward calculation is performed the level removed is printed as a reminder. (default iave=1, jave=1)

In effect a floating datum level is used and attention given to local anomalies rather than regional ones. This is the recommended mode of operation unless you have reason to believe your regional removal is perfect.

plenth Prism length in km measured from the profile to the end of the prism (total length is 2*plenth). Plenth will be overwritten by the individual prism lengths entered from the model file if at least one is nonzero. (Default = 0, for gravity this is a 2-d calculation; for magnetics 1000 kilometers is substituted as an approximation of 2-d.)

plotting parameters

There is an extensive set of defaults for proportioning areas for the graphs and cross-sectional view. Normally each plot defaults to a set fraction of the available plot area. When a scaling factor causes a view to exceed the plotter size the window is cut down to maintain the specified scaling; where viewport refers to the plotter and window refers to the data. Any combination of scaling and defaults may be specified.

iplotr Plot device type, this parameter is site dependent.
default=1 video, 5=HP-7580 series, 6=Versatec
The Versatec plotter has an internal default size of
72 x 20 inches.

vex Vertical exaggeration used when plotting the model. default=1

The scaling parameters default to zero which causes automatic proportioning of the remaining plotter area.

xscale model and graph horizontal scale in km/in.
zscale model vertical scale in km/inch. Default is set to xscale so an easy way to get a desired vertical exaggeration is to leave zscale blank and set 'vex'.
gscale gravity scaling in mGal/inch.
hscale magnetic scaling in nTesla/inch.

xlimit,ylimit in inches, is an alternate way of sizing the plot. These two parameters replace the physical plotting device size from which default scaling parameters are calculated. Scale and limit parameters should be mixed only with due care. Also note that internal switching of plotters causes scales and limits to be overwritten with new values.

The assigned window parameters are useful when graphs are to be overlayed, these will be overridden in case of viewport conflicts. Defaults are the min and max in the data.

xxx assigned x minimum, maximum window coordinates
zzz model depth window
ggg,hhh gravity and mag windows

The next 3 parameters are switches (0=off 1=on) that can also be accessed at runtime in the 'setup' function.

istatn (default 1) plots station locations on the cross-section
numbod (default 1) numbers the bodies
labelv (default 0) labels the vertices
iparsh (default 0) enables partial derivative plots in the inversion section. This parameter allows a user to look at the column vectors going into the linear system. Separate plots are generated for X and Z vertices, density, and susceptibility at three stages: calculated partials and weighted partials, and the final scaled system.

plot annotation parameters

title1 56 character label plotted top center.
title2 below title1
title3 below title2

titlx (20 char. default='kilometers') plotted under the graph.
titlz default='depth km', plotted to the left of the model.
titlg default='mGal', plotted to the left of the graph.
titlh default='nTesla', plotted to the left of the graph.

sizea axis label size (default=.08)
sizet axis title size for titlx,z,g,h (default=.1).
sizel titlel size (default=.15)

adelx horizontal axis ticking interval in kilometers (default about 50 ticks)
adelz model vertical axis (default about 15 ticks)
adelg gravity ticking interval in mgal (default about 20 ticks)
adelh magnetic in nTesla (20 ticks)

lintx,z,g,h (default=5) axis labelling intervals

PROGRAM INTERACTIVE COMMANDS

The function level commands are obtained after such preliminaries as command file input, model check, and synthetic data generation. For help, enter anything and SAKI will list the functions available. The program only recognizes the first two characters. To abort any operation enter garbage strings when the program is expecting a number.

Respond to the prompt "function :" with any of the following:

type allows terminal printing of model parameters. in response to the prompt "specify parm:" enter either
xbod for x body coordinates
zbod z body coordinates
rho densities
susc susceptiblity
pl1 near side prism coordinate (ie distance out from the profile)
pl2 far side prism coordinate
rems remanent magnetization strength
remi inclination of remanent magnetization vector
remd declination

Anything else will cause a prompting message. After selecting the parameter, the prompt "start,end:" is printed. Enter the starting and ending indices for the section you want to examine. A non-numeric character will cause a return to function level.
Example: specify parm:rho
start,end: 2 4 [request current densities for bodies
-.15 .25 0.0 2,3,and 4. User's input is underlined]

edit allows manual updating of the parameters listed for 'type'. After choosing the parm, the message "type,edit,return:" is printed; where 'type' is the same as the previous function, and 'return' returns to function level. The 'edit' command issues the prompt "enter n (indexl,valuel)..."; where
n is the number of pairs following
'index' is the index number of the parm to be changed
'value' is the new value

For instance, to move the x position of vertex 6 to 3.35, enter 1 6 3.35, at any time a non-numeric character will cause an abort and a prompt will be printed.

calculate places field values obtained from the basic forward routines into the array gcalc, and prints the dc removal and rms error values.

synthetic puts an array of field points into arrays xfield,zfield in response to the prompt "enter xo,dx,nobs,z:", where:
xo is starting x coordinate
dx spacing in km between points
nobs number of observations (grv+mag <= 400)
z level in km of the synthetic observations

The program then does a forward calculation and goes to the copy function, at this point array gcalc contains the calculated field values, and any real data which might be in arrays xfield,zfield is replaced by the new synthetic data.

copy puts gcalc into gobs in response to a yes answer.
One use is to set up a theoretical data set to test the inversion section of the program.

plot draws a graph of the field values and the cross-section of the model. The continuous curve represents the function calculated at the observation field points and the station symbols are plotted with either + (gravity) or a triangle (magnetics).

window allows a section of interest to be plotted. (default is the whole profile).

isolate allows up to 7 curves to be plotted each of which is the response of a selected group of bodies.

invert uses a modified Marquardt algorithm to change the linear parameters of density, susceptibility and the nonlinear parameters of x and z prism vertex locations. The program will not modify the lookup lists specifying the prism configurations or the remanent magnetization vector. (the lookup lists are the second line of the prism definition in the model file)
See a later section for details on the invert function.

setup allows resetting of runtime parameters, most can be initialized in the command file namelist. The three groups are datum type, magnetic direction, and plot parameters.
datum type sets a floating datum datum level for either gravity or magnetics (namelist equivalents - iave,jave)
magnetic direction sets Earth's field strength, inclination, declination, profile azimuth, and whether total field, x, y, or z component is calculated.
(namelist equivalents - efield,einc,edec,azmuth,compon)
plot parameters are the plotting device, model vertical exaggeration, new scaling or overall plot size (if not video terminal), and secondary switches istatn, numbod, and labelv.

magnetic allows internal switching from gravity to magnetic.
gravity allows internal switching from magnetic to gravity.
both turns on both gravity and magnetic calculations.
None of these three functions modify the existing data.

trend removes a linear trend from the observed data in response to a DC shift and slope entered by the user. A prompt will give DC and slope for two common cases: zeroing the ends of the profile, and fitting a least square line. The slope is in units of mGal/km or nT/km while the dc datum shift is in mGal or nT. The new observed curve can be saved by the output function.

noise salts either the calculated or observed curves with gaussian noise in reponse to a percent error entered by the user. A percent error of 1 will generate noise of about .1 mGal in gravity data with a 10 mGal range. If there is magnetic data present it will also be modified. The program will correlate the noise by local averages if desired.

output the model parameters as currently held in memory out to disk as the file 'model.out', will also output the observed gravity or magnetic field values or the forward calculated values as the file 'gfield.out' and 'hfield.out'. During a long modeling session it is a good idea to periodically output the model as a hedge against losing your updates.

exit program without writing either the model or observed or calculated data.

INVERSION FUNCTION

Experience gained from synthetic models is useful to get a feel for what is a well-posed inversion. In general fit the broad or deeper structures first with as simple a model as reasonable and work to finer details if the model constraints allow. For simple models, such as valley fill, with only a single interface; allow the bottom vertices to vary in depth while holding the horizontal positions fixed. If you have depth information (drill holes) for one or more locations, that information will constrain the interface to a unique solution. Models with more than one horizontal interface cannot be resolved and require outside constraints. These constraints come from other geophysical surveys, drill holes, and the geology of the area. Fault planes can be resolved for location, but the dip is dependent on the density constrast. The depth limit of resolution for an unconstrained feature is approximately equal to the lateral extent of the feature. The model should get more general with depth.

Simultaneous inversion of gravity and magnetic data can provide additional constraints because the two fields contribute different information. In general, the higher frequency content of magnetic data is useful for near surface, and gravity for deeper sources. The process of interactive modeling gives the user some information on which parts of the model are resolved. In many instances, some section of the model resists improvement and the geology has to be represented by another configuration. Reconverging a model is reasonably easy, test several alternatives.

The operator controls the inversion by selecting the free parameters which the program can vary. This decision is critical to the convergence of a relatively unconstrained model and requires a mix of art and science. One situation that always causes trouble is freeing the density and most of a body's vertices at once, since volume and density (susceptibility) are mathematically dependent. One way to begin an inversion is to enter your best guess for the model and if the response looks promising, invert on the density to get the amplitude in the ballpark, then lock the density and start inverting on the shape.

The parameters that can be iterated are the x and/or z vertice locations and the density or susceptibility with a maximum of 20 parameters free in one iteration. Once the free parameters are choosen several iterations may be performed without respecifying, normally three iterations is sufficient for one selection of parameters. For each iteration, the program will limit x excursions to 1/50 of the profile length, z to 1/10 the model thickness and density to .05 gm/cc, susceptibility is unconstrained.

Inversion example

Underlines indicate user input, refer to figure 4 for guidance.

```
function: invert
enter number of free x vertices: 4
enter indices : 39 42 35 40
enter number of free z vertices: 5
enter indices : 39 37 42 35 40
enter number of free densities : 0
enter number of free susceptibilites : 0

 active  gravity range (km)   36.00000        84.00000
 active magnetic range (km)   36.00000        84.00000
 weighted starting rms error     13.035

type parameter correlations ? yes
 a=
  2   4
  3   6  2
  4   3 11  8
  5  81  2  4  2
  6  26 16  3  2 19
  7   4 85  6  8  3 38
  8   4  8 64 31  2  2  7
  9   1  6 19 82  1  1  4  5
 b=   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
continue with the inversion ? yes

scaled eigenvalues
 1.0000  0.9751  0.7052  0.6891  0.6008
 0.3330  0.2956  0.2299  0.1495
cutoff      rms       angle      delta p
  1.0000   11.67       52.57      0.118
  0.5000   9.699       18.18      0.212
  0.2500   8.162       31.73      0.374
  0.1250   8.007       43.93      0.555
  0.1875   7.976       40.00      0.472
  0.2188   8.046       36.35      0.422
  0.3594   8.847       13.62      0.264
  0.2891   8.368       25.12      0.324
  0.3242   8.599       19.03      0.290
  0.3418   8.723       16.17      0.276
choosing eigenvalue cutoff  0.3418
  0.3418   8.723       16.17      0.276
```

```
parm      change        new value
x39       -0.5744         46.43
x42       -0.4176         59.58
x35        0.1158         66.33
x40       -0.1045         70.87
z39        0.2118        -1.146
z37       -0.2183        -1.732
z42       -0.1694        -1.368
z35     -9.2989E-02      -1.593
z40       -0.1608        -1.861

iteration 1, rms error is 8.723
percent improvement is 33.08

save new parameter values ? yes
another iteration ? no
```

Explanation

The objective is to improve the fit between coordinates 45 and 70 (figure 4). Body number 3 is low density volcanic fill and number 4 is a highly magnetic ring dike. Note the segment between vertices 41 and 36. This is an artifact of modeling and represents a change of magnetic susceptibility between bodies 5 and 4 while the density remains the same.

The interface between bodies 3 and 5 is allowed to vary in depth, and the positions of the dike and fill boundaries vary horizontally. First enter the free parameter indices, if these have been set during the current modeling session, the message "reset free parameters ?" is printed. Note that vertex 37 remains fixed in X because it would take a very large excursion to produce any change in mass distibution.

Next the active range data range is printed. CPU time is saved by limiting the the amount of data entering the inversion system.

The parameter correlations are printed as a triangular matrix with entries in percent correlation. They are numbered in the order of entry, ie. 1 is x39, 5 is z39. One hundred percent correlation indicates two parameters which are completely correlated and at least one eigenvalue will be very small. The inversion algorithm can handle this situation, but a better approach would be to exit to function level and change the parameter mix.

Next program will type out the system eigenvalues, and the damping factors tested should fall within the range between the largest and smallest eigenvalues. Should the linear system be unstable the damping factors will increase in value clamping down on the parameter excursions. The ratio between the largest and smallest eigenvalues should generally be less than 100. Up to 10 damping factors are tested and should one parameter have a large excursion or cross into a prism, appropriate warning messages are printed. RMS indicates the error between observed and calculated, ANGLE is the angle in parameter space between the proposed vector and the steepest descent vector, and DELTA P is a measure of the parameter change necessary (refer to figure 2).

The changes and new values of the parameters are printed out at the conclusion of the optimization and you may either accept or reject them. In the case the new values are no good, a no answer will leave the model the same and you can decide whether a different parameter mix or changes in the model at edit function level will get the model converging in the geologically reasonable direction.

Good luck

example usage 1

This first example includes both gravity and magnetic data along a profile directed south 45 degrees east. The command file contains data file and orientation information so that multiple runs can be made without parameter entry in the 'setup' function. Underlines indicate user input.

```
type profb.cmd
 &parms
 mfile='profb.m9', gfile='profb.grv', hfile='profb.mag'
 azmuth=135, efield=56500, einc=69, edec=18
 &

run saki
enter command filename : profb.cmd
number of vertices =  45
number of bodies   =   7
function: calc
removing       25.1     mgals from calculated gravity
removing      -20.2     nTesla from calculated magnetics
 gravity rms error      1.44
magnetic rms error      25.4
function: plot
```

Figure 3 is the basic starting plot that would be obtained on a video terminal. Each of the three diagrams occupies about one third the available screen size. The fitted curves represent continuous field values calculated at the data locations, and both observed values and station locations are plotted with either a plus (gravity) or a triangle (magnetics).

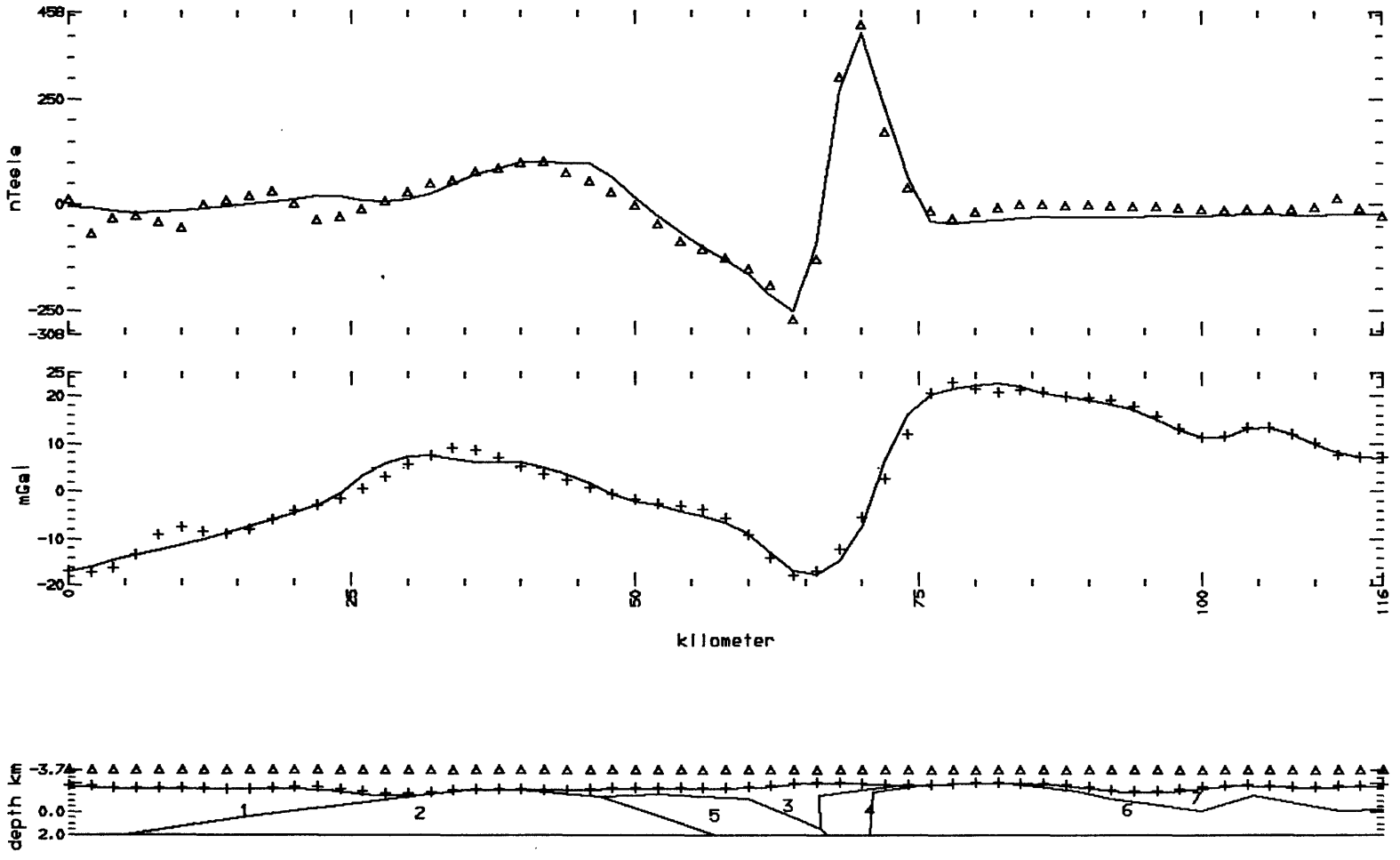


fig. 3 default plot

example usage 2

This example shows a typical plot generated during modeling. The window function sets the area of interest, station locations are suppressed, prism vertices are labeled, and the plot directed to a larger medium suitable for scratch work. Brackets indicate namelist parameter equivalents where appropriate.

```
function: window
enter profile min, max (0,0 to default) :
40 80                                    [ namelist equivalent is xxx ]

function: setup
setup parameters are:
datum type, magnetic direction
plot parameters, summary, or return

setup parameter: plot
 current plot device =          1
 do you want to change device ? y
 enter new device #: 0=Calcomp,1=Tek,5=HP,6=Vers
6                                                    [ iplotr ]
 enter 0 for an overall plot size
 or    1 for separate scaling factors
0
 enter maximum x,y plot size in inches
14 11                                                [ xlimit, ylimit ]
 (re)set vertical exaggeration and windows ? no      [ vex,zzz,ggg,hhh ]
 continue with secondary switches ? y
 plot station locations ? no                  [ istatn=0 ]
 plot body numbers ? yes                      [ numbod=1 ]
 plot vertex numbers ? yes                    [ labelv=1 ]
 enable partial derivative plots
 in the inversion function ? no               [ iparsh=0 ]

setup parameter: return
function: plot
```

Figure 4 is the resulting plot. Presentation style plots may be generated with the inclusion of titles and a specified scale that that overlays base maps (ie. xscale=6.35 km/in for map scale 1:250,000). Plot specifications remain until modified by the user. Scaling on video terminals always adjusts to give the maximum size plot, so returning to screen plots is easy.

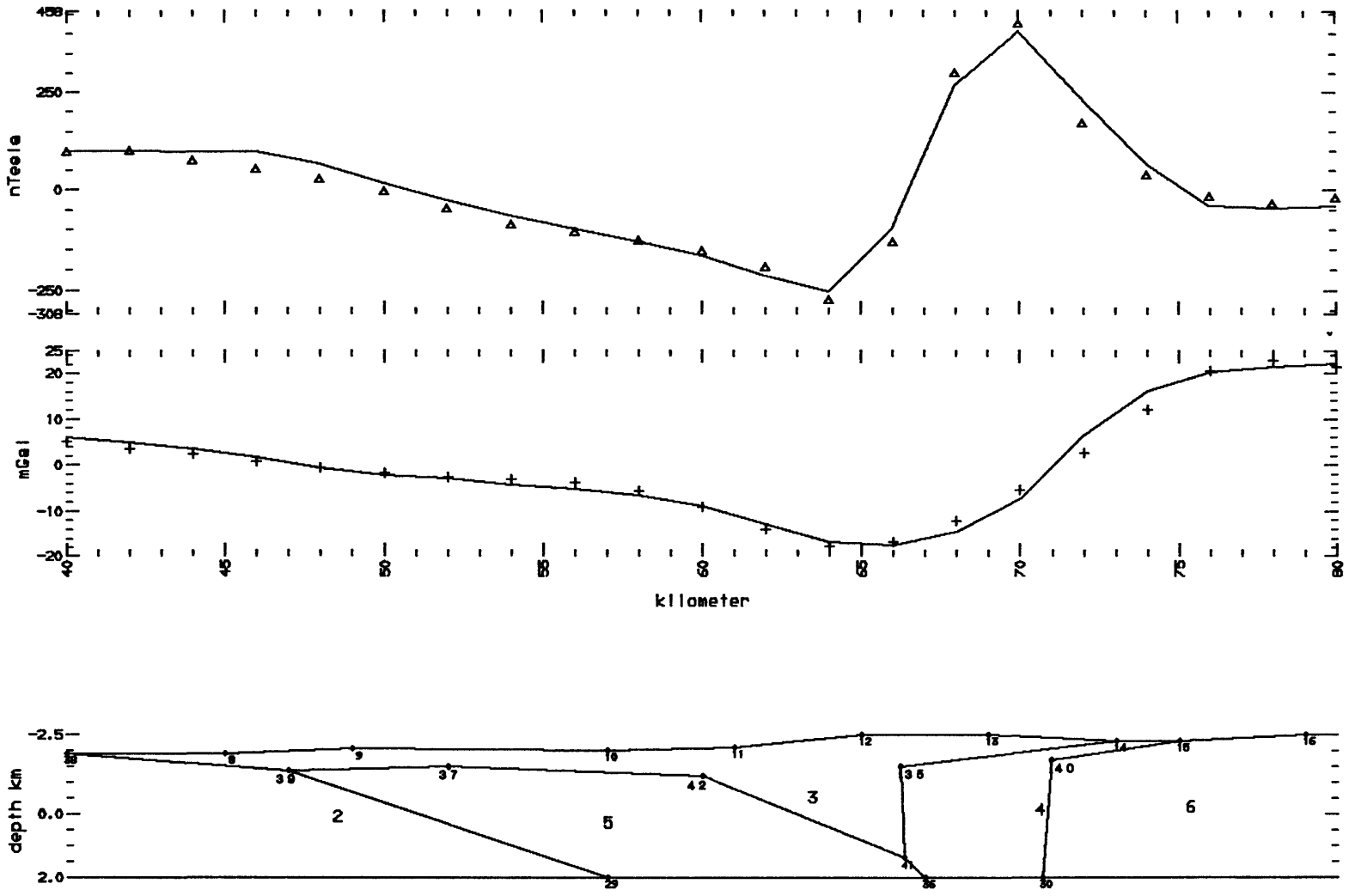


fig.4 expanded plot

example usage 3

The isolate function allows groups of body responses to be displayed separately to give the user an idea of what parameters are important.

```
 function: isolate
  enter number of curves to be plotted (max=7)
4
  enter nbody and body indices for curve          1
5  2 5 3 4 6          [summation of bodies 2,5,3,4, and 6]
  enter nbody and body indices for curve          2
1  5                  [body number 5]
  enter nbody and body indices for curve          3
1  3
  enter nbody and body indices for curve          4
1  4
```

Figure 5 is the result. The dashing patterns will vary with the site dependent plot driver system, here the curves also have drafted labels. As with other functions, non-numeric characters entered from the keyboard will cause the system to backtrack eventually to function level in case of errors.

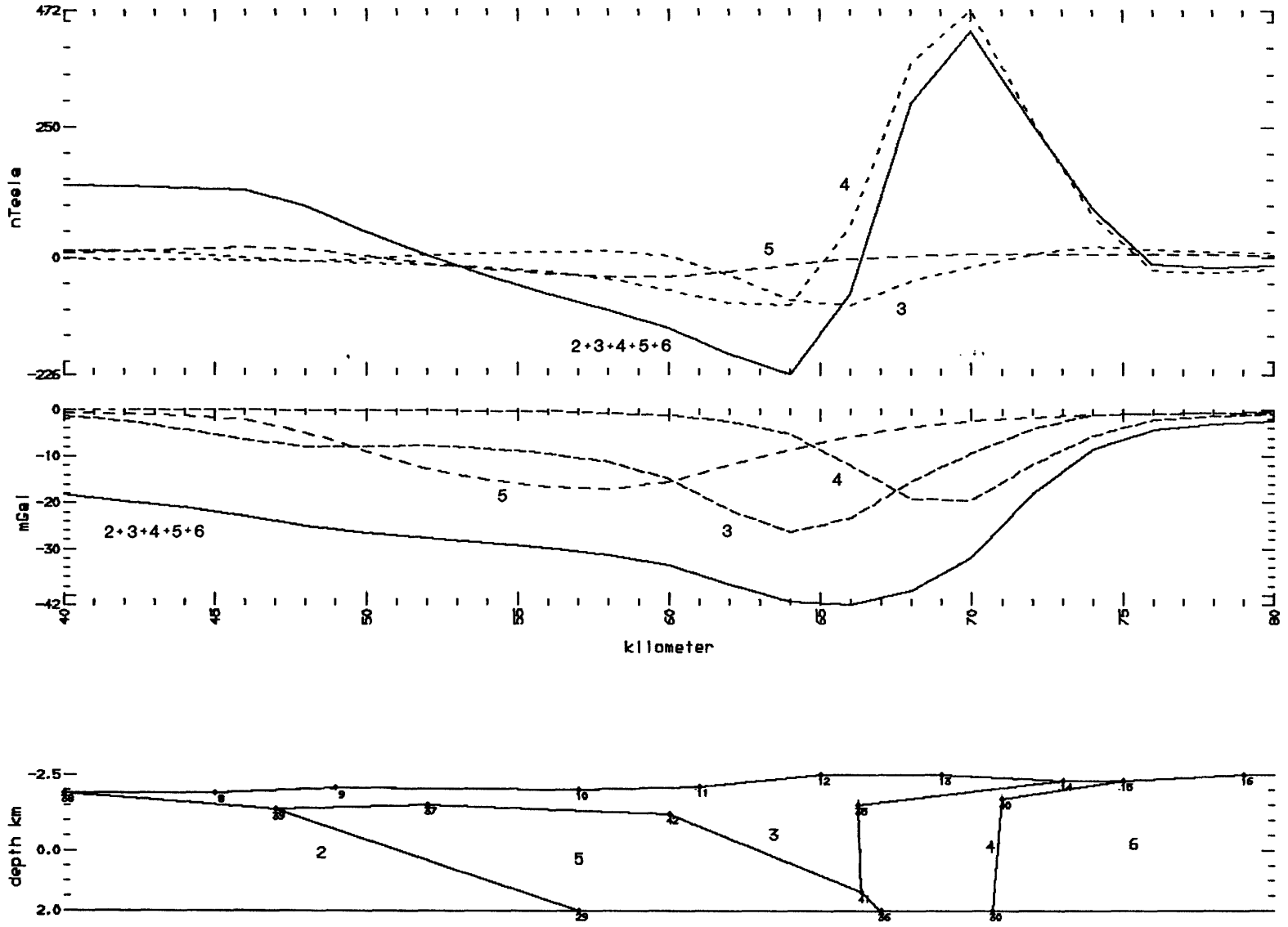


fig. 5 isolated body responses

References

Golub, G.H., Reinsch, C., 1970, Singular value decomposition and least-squares solutions, Num. Math., v. 14, 403-420.

Jackson, D.D., 1972, Interpretation of inaccurate, insufficient, and inconsistent data; Geophys. J. R. Astr. Soc., v. 28, 97-109.

Jupp, D., Vozoff, K., 1975, Stable iterative methods for the inversion of geophysical data; Geophys. J. R. Astr. Soc., v. 42, 957-976.

Lanczos, C., 1961, Linear differential operators, D. Van Nostrand Co., Lond.

Marquardt, D.W., 1963, An alogorithm for least-squares estimation of non-linear parameters, J. SIAM, v. 11, 431-441.

Penrose, R., 1955, A generalized inverse for matrices, Proc. Camb. Phil. Soc., v. 51, 406-413.

Rasmussen, R., Pedersen, L.B., 1979, End corrections in potential field modeling, Geophysical Prospecting, v. 27, 749-760.

Shuey, R.T., Pasquale, A.S., 1973, End corrections in magnetic profile interpretation, Geophysics, v. 38, 507-512.

Talwani, M., Worzel, J., Landisman, M., 1959, Rapid gravity computations for two-dimensional bodies with apllication to the Mendocino submarine fracture zone, J. Geophys. Res., v. 64, 49-59.

Appendix A

Plot system description

Plotting is performed by an unpublished subroutine package written by G.I. Evenden, USGS. SAKI uses the basic plotting functions:

```
PLTSET: device initialization
SCALE : scale data into plot units
LINE  : draw line segments
VCHAR : draw characters
ENDPT : close system
```

and two higher subroutines:

```
XAXIS : annotate horizontal axis
YAXIS : annotate vertical axis
```

PLTSET allows the user to select a plot device, and whether plot units will be inches or centimeters. The routine initializes the system and returns the size of the available plot area.
Example:

```
call pltset( iplotr, xlimit, ylimit, iunit)
```

where iplotr is an integer that selects the device
xlimit and ylimit are real variables specifying the plot area.
iunit is an integer selecting plot units. SAKI uses inches.

SCALE sets up a data area inside the plot area that is referenced in data units defined by the user (figure 6). Line segments are drawn using data units and clipped at the boundary of the data area. Character strings may be placed by refering to either plot or data units.

```
call scale (dxp, dyp, xp, yp, nopts, ierr)
```

where dxp and dyp are arrays with the minimum and maximum in data units such as kilometers or mGals.
xp and yp are in inches; xp(2), yp(2) are logarithmic switches.
nopts length of the xp and yp arrays.
ierr error parameter

LINE draws line segments between data points specified by two real arrays. Example:

```
call line ( x, y, n, icon, ipen )
```

where x and y are data point coordinates in data units.
n is the length of the x and y arrays.
icon equal 0 start new line, 1 continue from the previous line.

ipen select dashing pattern, range from 0 to 6.

VCHAR draws characters from four input modes. SAKI calls two of these modes from two internal subroutines SYMBOL and TEXT. SYMBOL plots a single character from the ASCII encoding sequence at locations specified from two coordinate arrays. Example:

```
call symbol ( x, y, n, ich, size, angle )
```

where x and y are two arrays of coordinates in data units

n	length of x and y arrays.
ich	is the number (base 10) of the ASCII character desired. The Evenden package defines the control characters 0 to 31 to be graphic symbols such as squares and triangles.
size	character size in inches.
angle	is counter-clockwise rotation angle in degrees.

TEXT plots a character string beginning at a location specified in plot units. Example:

```
call text ( x, y, string, size, angle )
```

where x and y are two real variables in plot units, specifying the first character location.

string	is a character string with length less than 100.
size	and angle are the same as for SYMBOL.

XAXIS annotates the horizontal boundary of the data area with tick marks and number labels.

Example: call xaxis (dxp, dyp, xp, del, ip, size, fmt, nfmt)

where

dxp	is the x data unit array described for SCALE.
dyp	y data unit array
xp	x plot unit array
del	ticking interval in data units
ip	label every ip'th tick mark
size	label size in inches
fmt	integer array containing the FORTRAN format specification of the labels.
nfmt	the number of characters plotted from the format. Example fmt = '(F8.2)' and nfmt = 7 would cause 123.49 to be plotted as 123.4.

YAXIS is similar to XAXIS

Example: call yaxis (dyp, dxp, yp, del, ip, size, fmt, nfmt)

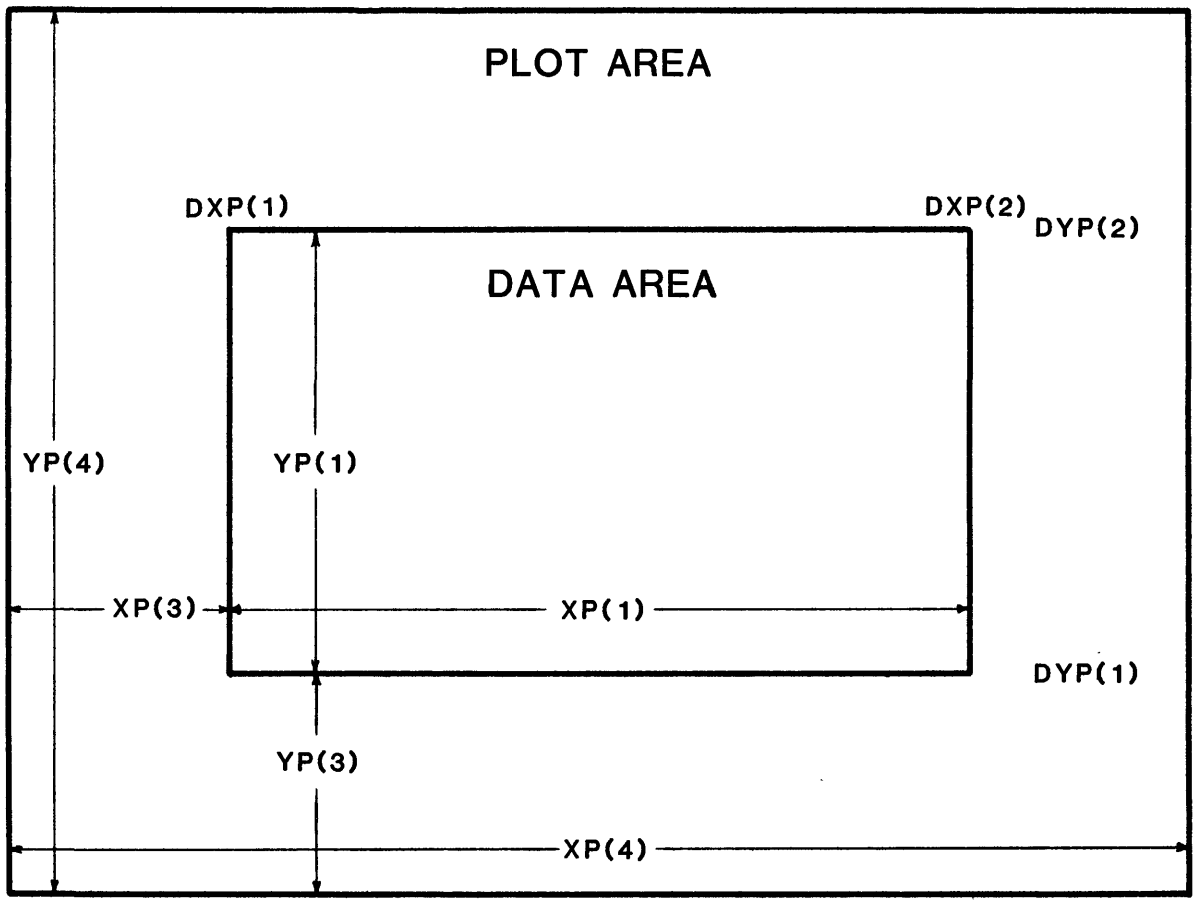


Figure 6. Data area layout

```
c                       Appendix B
c                  USGS Open File-Report 85-122
c
c                       program saki
c
c    Semi-Automatic MarQuardt Inversion of gravity and magnetic
c         profiles using singular value decomposition.
c
c  computer: DEC VAX
c  language: fortran 77
c  development history:
c  original gravity inversion program 10/80
c  2.5d gravity and magnetics 11/81
c  simultaneous inversion 6/83
c
c      Mike Webring
c  Br. of Geophysics, mail stop 964
c  US Geological Survey, box 25046
c  Denver Federal Center
c  Lakewood, CO  80225
c
      common /field/  nf(10),f(2000)
      common /model/  p(700)
      common /vertx/  iv(901)
      common /swork/  a(16820)
      common /parm1/  nvert,nbody,nobs,mobs,iswt,jswt,
     1               iave,jave,vex,istatn,iparsh,labelv,numbod
      common /parm2/  iofmt(14,4)
      common /magxyz/ ef(3),ev(3),tv(3),azmuth,icompn
      common /free1/  weight,rms1,iter,xk1,nfree(4),ifree(80)
      common /btags/  itag(450)
      common /plot1/  iplotr,sizea,sizet,sizel,
     1          title1,title2,title3,
     1          xscale,xxx(2),adelx,lintx,titlx,
     1          zscale,zzz(2),adelz,lintz,titlz,
     1          gscale,ggg(2),adelg,lintg,titlg,
     1          hscale,hhh(2),adelh,linth,titlh
      common /pltdev/ jplotr,xboard,yboard,xlimit,ylimit
      common /window/ xwind(2,4),ywind(2,4)
      common /vuport/ xview(4,4),yview(4,4)
      common /scal1/  onport(12)
c
      dimension xb(100),zb(100),rho(50),sus(50),pl1(50),pl2(50),
     1 rems(50),remi(50),remd(50)
      dimension icrn(500),ncrn(150),loca(150),ibord(100)
      dimension ipar(9),iparm(9)
c
      character    bfile*56, ifun*20,compon*1,iparm*4,prog*8
      character*20 fmtx,titlx,fmtz,titlz,fmtg,titlg,fmth,titlh
      character*56 test,cf,mfile,gfile,hfile,ifmtg,ifmth,ofmtg,ofmth
      character*56 title1,title2,title3
c
      equivalence (iswt,grvswt), (jswt,magswt)
      equivalence (p(1),xb), (p(101),zb), (p(201),rho), (p(251),sus),
```

```
     1 (p(301),pl1), (p(351),pl2), (p(401),rems), (p(451),remi),
     1 (p(501),remd)
      equivalence (ef(1),efield),(ef(2),einc),(ef(3),edec)
      equivalence (iv(1),icrn),    (iv(501),ncrn), (iv(651),loca),
     1           (iv(801),nbord), (iv(802),ibord)
      equivalence (iofmt(1,1),ifmtg), (iofmt(1,2),ifmth),
     1           (iofmt(1,3),ofmtg), (iofmt(1,4),ofmth)
c
      logical grvswt,magswt
c
      namelist /parms/ mfile,gfile,hfile,ifmtg,ifmth,ofmtg,ofmth,
     1 bfile,iave,jave,efield,einc,edec,azmuth,compon,plenth,
     1 istatn,iparsh,labelv,numbod,iplotr,vex,sizea,sizet,sizel,
     1 title1,title2,title3,
     1 xlimit,ylimit,
     1 xscale,xxx,adelx,lintx,titlx,
     1 zscale,zzz,adelz,lintz,titlz,
     1 gscale,ggg,adelg,lintg,titlg,
     1 hscale,hhh,adelh,linth,titlh
c
      data npar/9/,ipar/1,101,201,251,301,351,401,451,501/
      data iparm/'xbod','zbod','rho ','susc','pl1 ','pl2 ','rems',
     1 'remi','remd'/
      data maxfld/2000/
c
      idum=0
      ioutp=0
      test=' '
      mfile=' '
      bfile=' '
      gfile=' '
      hfile=' '
      ifmtg=' '
      ifmth=ifmtg
      ofmtg='(1p3g15.5)'
      ofmth=ofmtg
      compon=' '
      icompn=0
      efield=50000.0
      einc=90.
      edec=0.
      iave=1
      jave=1
      vex=1.0
      xxx(1)=0.0
      xxx(2)=0.0
      istatn=1
      iparsh=0
      labelv=0
      numbod=1
      do 1 i=1,2000
1     f(i)=0.
      do 2 i=1,700
2     p(i)=0.
```

```
      nxf=0
      nzf=0
      nrf=0
      call initp
c
109   type  110
110   format(' enter command filename :'$)
      read(5,37,end=109) cf
      if(cf.ne.test) then
        open (unit=9,file=cf,status='old',form='formatted',readonly)
        read(9,parms,end=111,err=111)
        if(bfile(1:1).ne.' ') mfile=bfile
        jplotr=iplotr
        go to 112
111     type *,' there is either a syntax error in the command file'
        type *,' or and invalid parameter.'
        type *,' do you want a list of valid namelist parameters ?'
        if(noyes(ldum).eq.1) then
          type *,' mfile,gfile,hfile,ifmtg,ifmth,ofmtg,ofmth'
          type *,' iave,jave,efield,einc,edec,azmuth,compon,plenth'
          type *,' istatn,iparsh,labelv,numbod'
          type *,' iplotr,xlimit,ylimit,vex,sizea,sizet,sizel'
          type *,' title*56,title2*56,title3*56'
          type *,' xscale,xxx(2),adelx,lintx,titlx*20'
          type *,' zscale,zzz(2),adelz,lintz,titlz*20'
          type *,' gscale,ggg(2),adelg,lintg,titlg*20'
          type *,' hscale,hhh(2),adelh,linth,titlh*20'
        endif
        close(9)
        stop
112     close(9)
      endif
      if(mfile.eq.test) then
        type  *,' model filename '
        read(5,37) mfile
      endif
      if( gfile.eq.test) then
        type *,' gravity data filename (optional)'
        read(5,37) gfile
      endif
      if(hfile.eq.test) then
        type *,' magnetic data filename (optional)'
        read(5,37) hfile
      endif
c
c  open data files, set function switches
      grvswt=.false.
      magswt=.false.
      open(unit=10,file=mfile,status='old',form='formatted',readonly)
      if(gfile.eq.test) go to 120
      grvswt=.true.
      open(unit=11,file=gfile,status='old',form='formatted',
     1 blank='zero',readonly)
120   if(hfile.eq.test) go to 121
```

```
      magswt=.true.
      open(unit=12,file=hfile,status='old',form='formatted',
     1 blank='zero',readonly)
c
c  read data
121   nobs=0
      if(.not.grvswt) go to 131
125   read(11,*,end=130)
      nobs=nobs+1
      go to 125
130   rewind 11
131   mobs=0
      if(.not.magswt) go to 141
135   read(12,*,end=140)
      mobs=mobs+1
      go to 135
140   rewind 12
c  place holders to avoid dimension errors
141   if(nobs.eq.0) nobs=1
      if(mobs.eq.0) mobs=1
c  data is stored in field common :xg,zg,go,gc,gerr, xh,zh,ho,hc,herr
      call initnf(nobs,mobs,nf,maxfld,ierr)
      if(ierr.eq.1) stop
      call datinp(nobs,f(nf(1)),f(nf(2)),f(nf(3)),
     1      mobs,f(nf(6)),f(nf(7)),f(nf(8)))
c
c  final initialization
      if (xxx(1).ge.xxx(2)) call initw(xxx,grvswt,magswt)
      call initl(plenth,pl,nbody,0)
      call cvc(compon,2)
      if(compon.eq.'x') icompn=1
      if(compon.eq.'y') icompn=2
      if(compon.eq.'z') icompn=3
      if(magswt) call initm(0)
      call outerb(nvert,nbody,a,a(301),a(601))
      call ckmodl(2,1,dum1,dum2,ierr1)
      if(grvswt) call ckmodl(1,nobs,f(nf(1)),f(nf(2)),ierr2)
      if(magswt) call ckmodl(1,mobs,f(nf(6)),f(nf(7)),ierr2)
      if(.not.grvswt .and. .not.magswt) go to 290
c
c  function branches
c
200   type  201
201   format(' function:')
      read(5,37,end=200) ifun
37    format(a20)
      call cvc(ifun,2)
      if(ifun(1:2).eq.'ca') go to 205
      if(ifun(1:2).eq.'in') go to 210
      if(ifun(1:2).eq.'wi') go to 220
      if(ifun(1:2).eq.'ty') go to 230
      if(ifun(1:2).eq.'pl') go to 240
      if(ifun(1:2).eq.'ed') go to 250
      if(ifun(1:2).eq.'ou') go to 260
```

```
      if(ifun(1:2).eq.'is') go to 270
      if(ifun(1:2).eq.'se') go to 280
      if(ifun(1:2).eq.'gr') go to 295
      if(ifun(1:2).eq.'ma') go to 295
      if(ifun(1:2).eq.'bo') go to 295
      if(ifun(1:2).eq.'sy') go to 300
      if(ifun(1:2).eq.'co') go to 310
      if(ifun(1:2).eq.'tr') go to 330
      if(ifun(1:2).eq.'no') go to 340
      if(ifun(1:2).eq.'ex') go to 999
      if(ifun(1:2).eq.'st') go to 999
      type  103
103   format(' help ? '$)
      if(noyes(1).eq.0) go to 200
      type  102
102   format(' functions: calculate, invert, plot, edit, type ',
     1 ' setup, window, synthetic, isolate body, trend removal ',/,
     1 ' noise, gravity, magnetic, both, output, exit')
      go to 200
c
c  forward calculation
205   call taldvr(nobs,f(nf(1)),f(nf(2)),f(nf(3)),f(nf(4)),f(nf(5)),
     1 rmsg,mobs,f(nf(6)),f(nf(7)),f(nf(8)),f(nf(9)),f(nf(10)),rmsh)
      if(grvswt) type 206,rmsg
206   format('  gravity rms error ',1pg13.3)
      if(magswt) type 207,rmsh
207   format(' magnetic rms error ',1pg13.3)
      go to 200
c
c  inversion
210   call nlctl(nobs,f(nf(1)),mobs,f(nf(6)))
      ngrv=0
      nmag=0
      if(grvswt) ngrv=nobs
      if(magswt) nmag=mobs
      no=ngrv+nmag
      go to 200
c
c  change operating window
220   type *,' enter profile min, max (0,0 to default) :'
      read(5,*,end=200,err=200) xxx
      if(xxx(1).ge.xxx(2)) call initw(xxx,grvswt,magswt)
      go to 200
c
c  type model parameter values
230   call getadr(npar,ipar,iparm,iad)
      call typdat(p(iad))
      go to 200
c
c  plot data and model
240   call pltdvr(ier)
      go to 200
c
c  edit model parameters
```

```
250   call getadr(npar,ipar,iparm,iad)
      call edtdat(p(iad))
      go to 200
c
c  output model and/or data
260   call modsav(ofmtg,ofmth)
      ioutp=1
      go to 200
c
c  plot isolated body curves
270   mgswt=0
      if(magswt) mgswt=1
      if(grvswt.and.magswt) mgswt=-1
      call setwin(mgswt,xxx,zzz,ggg,hhh)
      call isodvr
      go to 200
c
c  change parameters
280   call setup
      go to 200
c
c  switch calculations
290   type 291
291   format(' activate gravity, magnetic, or both :')
      read(5,37,end=290) ifun
      call cvc(ifun,2)
295   grvswt=.false.
      magswt=.false.
      if(ifun(1:2).eq.'gr') grvswt=.true.
      if(ifun(1:2).eq.'ma') magswt=.true.
      if(ifun(1:2).eq.'bo') grvswt=.true.
      if(ifun(1:2).eq.'bo') magswt=.true.
      iter=1
      if(.not.grvswt .and. .not.magswt) go to 290
      if(magswt) call initm(0)
      if(grvswt .and. nobs.le.1) go to 300
      if(magswt .and. mobs.le.1) go to 300
      go to 200
c
c  synthetic data
c  field values are stored xg,zg,go,gc,gerr, xh,zh,ho,hc,herr
300   if(.not.grvswt) go to 305
      if(nobs.gt.1) then
        type 411
411     format(' change gravity locations ?'$)
        if(noyes(ldum).eq.0) go to 305
      endif
      type  301
301   format( ' enter synthetic gravity   xo,dx,nobs,z :'$)
      read(5,*,end=305,err=305) x,dx,nobs,z1
      if(nobs.le.0) nobs=1
c  move magnetic data
       iad=nf(6)
       do 400 i=1,5*mobs
```

```
       a(i)=f(iad)
400    iad=iad+1
       do 401 i=1,5*nobs
401    f(i)=0.0
       iad=5*nobs+1
       do 402 i=1,5*mobs
       f(iad)=a(i)
402    iad=iad+1
      call initnf(nobs,mobs,nf,maxfld,ierr)
      if(ierr.eq.1) go to 300
      type *,'is the z level a constant clearance distance ?'
      if (noyes(ldum).eq.0) then
        do i = 0, nobs-1
          f(nf(1)+i) = x
          f(nf(2)+i) = zl
          x = x + dx
        enddo
      else
        call drpsur(x,dx,nobs,zl, f(nf(1)),f(nf(2)) )
      endif
c  enter synthethic mag locations
305   if(.not.magswt) go to 309
      if(mobs.gt.1) then
        type 406
406     format(' change magnetic locations ?'$)
        if(noyes(1)) 309,309,410
      endif
410   type  306
306   format( ' enter synthetic magnetic  xo,dx,nobs,z :'$)
      read(5,*,end=309,err=309) x,dx,mobs,zl
      if(mobs.le.0) mobs=1
      call initnf(nobs,mobs,nf,maxfld,ierr)
      if(ierr.eq.1) go to 410
      type *,'is the z level a constant clearance distance ?'
      if (noyes(ldum).eq.0) then
        do i = 0, mobs-1
          f(nf(6)+i) = x
          f(nf(7)+i) = zl
          x = x + dx
        enddo
      else
        call drpsur(x,dx,mobs,zl, f(nf(6)),f(nf(7)) )
      endif
309   continue
      call taldvr(nobs,f(nf(1)),f(nf(2)),f(nf(3)),f(nf(4)),f(nf(5)),
     1 rmsg,mobs,f(nf(6)),f(nf(7)),f(nf(8)),f(nf(9)),f(nf(10)),rmsh)
c
c  copying function
310   if(.not.grvswt .or. nobs.le.1) go to 314
      type  311
311   format(' copy calculated gravity into observed ?'$)
      if(noyes(1).eq.0) go to 314
      do 313 i=0,nobs-1
313   f(nf(3)+i)=f(nf(4)+i)
```

```
c
314   if(.not.magswt .or. mobs.le.1) go to 200
      type  315
315   format(' copy calculated magnetics into observed ?'$)
      if(noyes(1).eq.0) go to 200
      do 316 i=0,mobs-1
316   f(nf(8)+i)=f(nf(9)+i)
      go to 200
c
c  trend removal
330   if(grvswt) then
      call trend(f(nf(1)),f(nf(3)),nobs,dc,dy)
      call ammi(nobs,f(nf(3)),gmin,gmax,mn,mx,1)
      type  335,gmin,gmax
335   format(' residual data is stored in the observation array.',/,
     1 ' the new gravity data has  a min/max of :',1p2e15.5)
      endif
      if(magswt) then
      call trend(f(nf(6)),f(nf(8)),mobs,dc,dy)
      call ammi(mobs,f(nf(8)),gmin,gmax,mn,mx,1)
      type  336,gmin,gmax
336   format(' residual data is stored in the observation array.
     1 ' the new magnetic data has  a min/max of :',1p2e15.5)
      endif
      go to 200
c
c  add noise to data
340   call adnois(nf,f,a,grvswt,magswt)
      go to 200
c
999   close(10)
      if(ioutp.eq.0) then
        type *,' write the current model to disk ?'
        if(noyes(1dum).eq.1) call modsav(ofmtg,ofmth)
      endif
      stop
      end
c*********************************************************************
      subroutine datinp(nobs,xg,zg,go, mobs,xh,zh,ho)
c  example model contains vertex coordinate pairs followed by
c  2 line body specifications, the first line is physical parameters
c  and the second line a lookup list which specifies vertex number in
c  clockwise order around the body.
c  -2 1  2 1  2 2  -2 2
c  0 1 0 3  0 -1
c  <<<<
c  .2 1.e-3 20. -20. 0.e-5 40. 0.
c  1 5 6 4 <
c  -.2 -1.e-3 20. -20. 0.e-5 60. 0.
c  5 2 3 6 <
      common /model/ xb(100),zb(100),rho(50),sus(50),
     1 pl(50,2),rem(50,3)
      common /vertx/ icrn(500),ncrn(150),loca(150)
      common /parml/ nvert,nbody,nobsl,mobsl,grvswt,magswt,
```

```
     1 iave,jave,vex,idepth,iparsh,indiv
       common /parm2/ fmtg,fmtm
       common /swork/ tx(400),tz(400),tg(400),link(400)
       common /magxyz/ ef(3),ev(3),tv(3),azmuth
       dimension xg(1),zg(1),go(1),xh(1),zh(1),ho(1)
       logical grvswt,magswt,nofmt
       character*56 fmtg,fmtm
       character*80 line,blank
       blank=' '
c
       iunit=10
       call countv(nvert,iunit,iend)
       if(iend.eq.1) stop ' datinp: eof1 model file'
       nbody=0
5      nbody=nbody+1
       read(iunit,*,end=6)
       call countv(ncrn(nbody),iunit,iend)
       if(iend.eq.1) go to 6
       go to 5
6      nbody=nbody-1
       rewind 10
       nvert=nvert/2
       type 10,nvert,nbody
10     format(' number of vertices =',i4,/,
     1 ' number of bodies   =',i4)
c  read vertex coordinates
       read(10,*,err=66) (xb(i),zb(i),i=1,nvert)
c  skip blank lines, find body list delimiter
       do i=1,5
         read(10,*,err=11) delimt
       enddo
c  read physical property and vertex lookup lists
11     j=1
       ivert=0
       is=1
       do 20 ibody=1,nbody
       loca(ibody)=is-1
       ie=loca(ibody)+ncrn(ibody)
c  blank line before next body spec?
       read(10,13) line
13     format(a)
       if (line.ne.blank) backspace (10)
       read(10,*,err=77,end=99) rho(ibody),sus(ibody),pl(ibody,1),
     1 pl(ibody,2),rem(ibody,1),rem(ibody,2),rem(ibody,3)
       read(10,*,err=88,end=99) (icrn(i),i=is,ie)
       is=ie+1
20     continue
c
c  get body tags
       call bodyid(nvert,nbody)
c
c  read field observations
       if(.not.grvswt) go to 40
       nofmt=.true.
```

```
      if( fmtg(1:1).eq.'(' ) nofmt=.false.
      do 30 i=1,nobs
      if(nofmt) then
      read(11,*) xg(i),zg(i),go(i)
      else
      read(11,fmtg) xg(i),zg(i),go(i)
      endif
30    continue
40    if(.not.magswt) go to 160
      nofmt=.true.
      if( fmtm(1:1).eq.'(' ) nofmt=.false.
      do 50 i=1,mobs
      if(nofmt) then
      read(12,*) xh(i),zh(i),ho(i)
      else
      read(12,fmtm) xh(i),zh(i),ho(i)
      endif
50    continue
c
c  sort gravity observations in x
160   if(nobs.eq.0 .and. mobs.eq.0) return
      do 161 i=2,nobs
      if(xg(i).lt.xg(i-1)) go to 165
161   continue
      return
165   call shsort(nobs,xg,link,0)
      do 166 i=1,nobs
      tx(i)=xg(i)
      tz(i)=zg(i)
166   tg(i)=go(i)
      do 167 i=1,nobs
      m=link(i)
      xg(i)=tx(m)
      zg(i)=tz(m)
167   go(i)=tg(m)
      return
66    type *,' datinp: vertex read error'
      return
77    type *,' datinp: phys-prop input error body number',ibody
      return
88    type *,' datinp: look-up read error body number',ibody
      return
99    type *,' datinp: eof in body definition'
      return
      end
c**********************************************************************
      subroutine countv(ival,iunit,iend)
c  count words delimited by spaces or commas
c  return when non-alpanumeric character begins word
      character a*200
      iend=0
      ival=0
10    a=' '
      read(iunit,20,end=99) a
```

```
20    format(a200)
30    na=leftj(a)
      if(na.eq.0) go to 10
      if(a(1:1).eq.',') then
      a(1:na-1)=a(2:na)
      a(na:na)=' '
      go to 30
      endif
      ich=ichar(a(1:1))
      if(ich.le.42 .or. ich.ge.123) return
      if(ich.ge.58 .and. ich.le.64) return
      if(ich.ge.91 .and. ich.le.96) return
      mb=index(a,' ')
      mc=index(a,',')
      m=mb
      if(mc.gt.0 .and. mc.lt.mb) m=mc
      ival=ival+1
      na1=na-m+1
      a(1:na1)=a(m:na)
      do 40 i=na1+1,200
40    a(i:i)=' '
      go to 30
99    iend=1
      return
      end
c**********************************************************************
      subroutine bodyid(nvert,nbody)
c  get the laundry tag at the end of the body lookup list
      common /swork/ a(2000)
      common /btags/ itag(450)
      character*36 tag(50)
      character line*80,blank*80,ischar*1
      data blank/' '/
      equivalence (itag,tag)
      logical alphan
      len1=80
      lentag=36
      rewind 10
      read(10,*,err=5,end=99) (a(i),i=1,2*nvert)
      do i=1,5
        read(10,*,err=5,end=99) delimt
      enddo
5     do 100 ibody=1,nbody
      tag(ibody)=' '
      read(10,*,err=99,end=99)
10    line=' '
      read(10,20,err=99,end=99) line
      if (line.eq.blank) go to 10
20    format(a80)
      do 50 ich=1,len1
      if(.not.alphan(line(ich:ich))) then
        do 40 is=ich+1,len1
        ischar=line(is:is)
        if(.not.(alphan(ischar)) .or. ischar.eq.' ') go to 40
```

```
        ie = is + (lentag-1)
        if(ie.gt.lenl) ie=lenl
        tag(ibody)=line(is:ie)
        go to 100
40      continue
      endif
50    continue
      go to 10
100   continue
99    return
      end
c**********************************************************************
      logical function alphan(ch)
c  is 'ch' alpha-numeric ?
      character ch*(1)
      alphan=.false.
      i=ichar(ch)
      if(i.eq.32 .or.  i.eq.43)  go to 10
      if(i.eq.45 .or.  i.eq.46)  go to 10
      if(i.ge.48 .and. i.le.57)  go to 10
      if(i.ge.65 .and. i.le.90)  go to 10
      if(i.ge.97 .and. i.le.122) go to 10
      return
10    alphan=.true.
      return
      end
c**********************************************************************
      subroutine outerb(nvert,nbody,idupl,is,ie)
c   assemble an array defining the outer boundary
c   of the model for checking vertex relations.
      common /vertx/ icrn(500),ncrn(150),loca(150),nbord,ibord(100)
      dimension idupl(1),is(1),ie(1)
      data maxs/300/,maxb/100/
      nbord=0
      ic=1
      do 10 j=1,nbody
      do 10 i=1,ncrn(j)
      idupl(ic)=0
      is(ic)=icrn(loca(j)+i)
      il=i+1
      if(i.eq.ncrn(j)) il=1
      ie(ic)=icrn(loca(j)+il)
      if((is(ic).lt.1 .or. is(ic).gt.nvert) .or.
     1  (ie(ic).lt.1 .or. ie(ic).gt.nvert)) then
        type *,' outerb:  lookup list for body ',j
        type *,' contains an entry < zero or >', nvert, '(nvert)'
        type *,' corners',i,il,' point to vertices',is(ic),ie(ic)
        stop
      endif
      ic=ic+1
      if(ic.gt.maxs) go to 999
10    continue
      nseg=ic-1
c
```

```
      do 50 iseg=1,nseg
      iv1=is(iseg)
      iv2=ie(iseg)
      do 20 itst=iseg+1,nseg
      if(iv1.eq.is(itst) .and. iv2.eq.ie(itst)) go to 15
      if(iv1.eq.ie(itst) .and. iv2.eq.is(itst)) go to 15
      go to 20
15    idupl(iseg)=idupl(iseg)+1
      idupl(itst)=idupl(itst)+1
20    continue
50    continue
c
      icount=0
      do 60 i=1,nseg
      if(idupl(i).gt.1) then
      type  61,is(i),ie(i)
61    format(' segment', 2i4,' found more than twice')
      icount=icount+1
      if(icount.gt.5) stop
      endif
      if(idupl(i).eq.0) idupl(i)=-1
60    continue
c
      ipass=0
      do 70 ifirst=1,nseg
      if(idupl(ifirst).ge.0) go to 70
      ibord(1)=is(ifirst)
      ibord(2)=ie(ifirst)
      nbord=2
      go to 71
70    continue
c
71    ifind=ifirst+1
      ipass=ipass+1
      if(ipass.gt.nseg) go to 777
72    if(idupl(ifind).ge.0) go to 75
      if(is(ifind).ne.ibord(nbord)) go to 75
      nbord=nbord+1
      if(nbord.gt.maxb) go to 999
      ibord(nbord)=ie(ifind)
      if(ibord(nbord).eq.ibord(1)) go to 80
      idupl(ifind)=0
      go to 71
75    ifind=ifind+1
      if(ifind.gt.nseg) go to 71
      go to 72
80    nbord=nbord-1
      return
777   type *,' outerb: did not close polygon'
      type *,' current vertex array'
      type *,(ibord(i),i=1,nbord)
999   type  998
998   format(' no model boundary check will be done')
      return
```

```
      end
c*********************************************************************
      subroutine ckmodl(ifun,npt,xf,zf,ierr)
c  ifun=0 check both terrain and prisms
c  ifun=1 check terrain, ifun=2 check prism twists
      common /parm1/ nvert,nbody,nobs,mobs,grvswt,magswt
      common /model/ xb(100),zb(100)
      common /vertx/icrn(500),ncrn(150),loca(150),nbord,ibord(100)
      common /swork/ xf1(400),zf1(400),x(100),z(100)
      dimension lint(2),xf(npt),zf(npt)
      logical grvswt,magswt,inside,member,planar
      ierr=0
      nn=loca(nbody)+ncrn(nbody)
      if(ifun.eq.2) go to 35
c
c   check for vertices above terrain (observations)
      do 20 i=1,npt
      xf1(i)=xf(i)
20    zf1(i)=zf(i)
      m=npt+2
      xf1(npt+1)=xf1(npt)
      xf1(m)=xf1(1)
      zf1(npt+1)=-100.
      zf1(m)=-100.
      do 30 i=1,nvert
      if(.not.member(i,icrn,nn)) go to 30
      if(inside(m,xf1,zf1,xb(i),zb(i))) type  25,i
25    format(' vertex',i3,' is above observations')
30    continue
      if(ifun.eq.1) return
c
35    do 50 j=1,nbody
      if(planar(ncrn(j),xb,zb,icrn(loca(j)+1),lint)) go to 50
      ierr=1
      type *,' ckmodl>> body number ',j,' has an intersection'
      type *,'       between line segments', lint(1), 'and', lint(2)
50    continue
      if(planar(nbord,xb,zb,ibord,lint)) return
      ierr=1
      type *,' ckmodl>> the main model outer boundary has an'
      type *,'          intersection between line segments'
      type *,lint(1), 'and', lint(2)
      return
      end
c*********************************************************************
      subroutine getadr(npar,ipar,iparm,iad)
      dimension ipar(npar),iparm(npar)
      character what*8,iparm*4
30    type  31
31    format(' specify parameter :'$)
      read(5,5,end=30) what
5     format(a8)
      call cvc(what,2)
      do 32 i=1,npar
```

```
      if(what(1:4).eq.iparm(i)) go to 34
32    continue
      type  33,iparm
33    format(' model parameters available (enter 4 char.)',/,
     1 10(1x,a4,',') )
      go to 30
34    iad=ipar(i)
      return
      end
c**********************************************************************
      subroutine typdat(a)
      dimension a(1)
1     type  2
2     format(' start,end :'$)
      read(5,*,end=20,err=20) is,ie
      if(is.le.0) return
      if(ie-is+1 .gt. 50) then
        type *,' limiting output to 50 values'
        ie=is+49
      endif
      type  10,(a(i),i=is,ie)
10    format(5g13.5)
      go to 1
20    return
      end
c**********************************************************************
      subroutine edtdat(a)
      dimension a(1)
      character func*16
      ierr=0
1     type  2
2     format(' ed, ty, ret : '$)
      read(5,3,end=1) func
3     format(a16)
      call cvc(func,2)
      nch=leftj(func)
      if(func(1:1).ne.'e') go to 10
        type 5
5       format(' enter n pairs (index1, value1)...'$)
        read(5,*,end=1,err=1) n,(i,a(i),m=1,n)
        go to 1
10    if(func(1:1).ne.'t') go to 15
        call typdat(a)
        go to 1
15    if(func(1:1).eq.'r') return
        ierr=ierr+1
        if(ierr.gt.3) return
        type 20
20      format(' edit functions = edit, type, return')
      go to 1
      end
c**********************************************************************
      subroutine modsav(ofmtg,ofmtm)
      common /field/  nf(10),f(2000)
```

```
      common /parm1/  nvert,nbody,nobs,mobs,grvswt,magswt
      common /model/  xb(100),zb(100),rho(50),sus(50),
     1               pl(50,2),rem(50,3)
      common /vertx/  icrn(500),ncrn(150),loca(150)
      common /swork/  a(8800)
      common /btags/  itag(450)
      logical grvswt,magswt
      character ofmtg*56,ofmtm*56,ians*16,fmt1*24
      character*36 tag(50),blk20
      equivalence (itag,tag)
      blk20=' '
37    format(a16)
c
      open(unit=12,file='model.out',form='formatted',
     1 carriagecontrol='list',status='new')
      ic=0
      do 2 i=1,nvert
      ic=ic+1
      a(ic)=xb(i)
      ic=ic+1
2     a(ic)=zb(i)
      n2=2*nvert
      write(12,5,err=8) (a(i),i=1,n2)
c  5 lines (20 vertices) then a blank line
5     format( 5( 5(1x, 4(f8.2,1x,f7.3,2x),/) ,/) )
      go to 9
8     type *,' modsav:lost some vertices due to formatting error'
c
9     write(12,10,err=15)
10    format(' >> body specifications')
c
15    do 50 i=1,nbody
      write(12,20,err=21) rho(i),sus(i),pl(i,1),pl(i,2),
     1 rem(i,1),rem(i,2),rem(i,3),i
20    format((1x,f6.3), (1x,1pg11.3), 0p2(1x,f8.1),
     1 (1x,1pg11.3), 0p2(1x,f6.1), ' <<phys_prop',i2)
c
21    nleft=ncrn(i)
      if(nleft.gt.200) go to 50
      iadr=loca(i)
      is=1
25    if(nleft.le.10) go to 35
      write(12,30) (icrn(i2),i2=iadr+is,iadr+is+9)
30    format(10i4)
      nleft=nleft-10
      is=is+10
      go to 25
c
35    write(fmt1,40,err=45) nleft
40    format( '(', i2, 'i4,'' < '',a36)')
      if(tag(i)(1:36).eq.blk20(1:36)) then
        write(tag(i),43) i
43      format('body',i3)
      endif
```

```
      write(12,fmt1,err=45)
     1 (icrn(i2),i2=iadr+is,iadr+ncrn(i)),tag(i)
      go to 50
45    type *,' modsav:lost a lookup list for body',i
50    continue
      close(12)
c
c  output field values
      type 51
51    format(' output field values ?')
      if(noyes(ldum).eq.0) return
55    type 60
60    format(' calculated or observed :')
      read(5,37,end=55) ians
      call cvc(ians,2)
      ical=-1
      if(ians(1:1).eq.'o') ical=0
      if(ians(1:1).eq.'c') ical=1
      if(ical.eq.-1) go to 55
c
      if(grvswt) then
      open(unit=13,file='gfield.out',form='formatted',
     1 carriagecontrol='list',status='new')
      n1=nf(1)
      n2=nf(2)
      n3=nf(3+ical)
      do 70 i=0,nobs-1
70    write(13,ofmtg,err=71) f(n1+i),f(n2+i),f(n3+i)
71    close(13)
      endif
c
      if(magswt) then
      open(unit=13,file='hfield.out',form='formatted',
     1 carriagecontrol='list',status='new')
      n1=nf(6)
      n2=nf(7)
      n3=nf(8+ical)
      do 90 i=0,mobs-1
      write(13,ofmtm,err=91) f(n1+i),f(n2+i),f(n3+i)
90    continue
91    close(13)
      endif
      return
      end
c**********************************************************************
      subroutine taldvr(nobs,xg,zg,go,gc,gerr,rmsg,
     1 mobs,xh,zh,ho,hc,herr,rmsm)
c   basic forward calculation, the talwani driver cycles through
c   the body array, removes a dc level, and figures rms error.
      common /parm1/ nvert,nbody,nobs1,mobs1,grvswt,magswt,
     1 iave,jave,vex,idepth,iparsh,indiv
      common /swork/ xb1(100),zb1(100)
      common /model/ xb(100),zb(100),rho(50),sus(50),p1(50,2),
     1 rem(50,3)
```

```
      common /vertx/ icrn(500),ncrn(150),loca(150),nbord,ibord(100)
      dimension xg(1),zg(1),go(1),gc(1),gerr(1),
     1 xh(1),zh(1),ho(1),hc(1),herr(1)
      logical inside,grvswt,magswt
c
      if(grvswt) then
      do 1 i=1,nobs
1     gc(i)=0.0
      rmsg=1.e30
      endif
      if(magswt) then
      do 2 i=1,mobs
2     hc(i)=0.0
      rmsm=1.e30
      endif
      call ckmodl(2,nobs,xg,zg,ierr)
c       if(grvswt) call ckmodl(1,nobs,xg,zg,ierr)
      if(magswt) call ckmodl(1,mobs,xh,zh,ierr)
      if(ierr.eq.1) return
c
c  check for overlapping bodies
      ic=1
      do 100 j=1,nbody
      do 5 i=1,ncrn(j)
      xbl(i)=xb(icrn(loca(j)+i))
5     zbl(i)=zb(icrn(loca(j)+i))
      do 6 k=1,nvert
      do 9 l=1,ncrn(j)
      if(k.eq.icrn(loca(j)+l)) go to 6
9     continue
      if(.not.inside(ncrn(j),xbl,zbl,xb(k),zb(k))) go to 6
      type  7,k,j
7     format(' vertex',i3,' is inside body',i3)
      return
6     continue
c
c  call forward routines
      jl=j
      if(grvswt) call pickmg(0,jl,rho(j),ncrn(j),xbl,zbl,
     1 nobs,xg,zg,gc)
      if(magswt)  call pickmg(1,jl,sus(j),ncrn(j),xbl,zbl,
     1 mobs,xh,zh,hc)
100   continue
c
c  gravity errors
      if(.not.grvswt) go to 30
      if(iave.eq.0) go to 19
      ave=0.0
      do 16 i=1,nobs
16    ave=ave+(go(i)-gc(i))
      ave=ave/float(nobs)
      type 17,ave
17    format(' removing',1pg15.3,' mgals from calculated gravity')
      do 18 i=1,nobs
```

```
18    gc(i)=gc(i)+ave
19    errt=0.0
      do 20 i=1,nobs
      gerr(i)=go(i)-gc(i)
20    errt=errt+gerr(i)*gerr(i)
      rmsg=sqrt(errt/float(nobs))
c
c  magnetic errors
30    if(.not.magswt) return
      if(jave.eq.0) go to 39
      ave=0.0
      do 36 i=1,mobs
36    ave=ave+(ho(i)-hc(i))
      ave=ave/float(mobs)
      type 37,ave
37    format(' removing',1pg15.3,' nTesla from calculated magnetic
      do 38 i=1,mobs
38    hc(i)=hc(i)+ave
39    errt=0.0
      do 40 i=1,mobs
      herr(i)=ho(i)-hc(i)
40    errt=errt+herr(i)*herr(i)
      rmsm=sqrt(errt/float(mobs))
      return
      end
c**********************************************************************
      subroutine pickmg(mgswt,ibody,rho,nvert,xb,zb,nfld,xf,zf,pf)
c  switching routine for the individual forward algorithms
c  rho is the passed linear parameter (density or susceptibility)
c  prism lengths and remanent magnetic values are passed thru common
c  xb,zb is one body coordinate array
c  xf,zf array of locations to be used
      common /model/ dum(300),pl(50,2),rf(50,3)
      common /magxyz/ ef(3),ev(3),tdir(3),azmuth
      dimension yl(2),vm(3),xb(1),zb(1),xf(1),zf(1),pf(1)
      data d2r/1.745329e-2/
      yl(1)=pl(ibody,1)
      yl(2)=pl(ibody,2)
      if(mgswt.eq.0) go to 10
      rinc=d2r*rf(ibody,2)
      rdec=d2r*(rf(ibody,3)-azmuth)
      rl=cos(rdec)*cos(rinc)
      rm=sin(rdec)*cos(rinc)
      rn=sin(rinc)
c  convert emu/cc to gamma
      rem=rf(ibody,1)*1.e5
c  polarization vector
      vm(1)=rho*ef(1)*ev(1)+rem*rl
      vm(2)=rho*ef(1)*ev(2)+rem*rm
      vm(3)=rho*ef(1)*ev(3)+rem*rn
      ylen=abs(yl(1))
      if(ylen.eq.0.0) ylen=1000.
      call mag2hd(vm,tdir,ylen,nvert,xb,zb,nfld,xf,zf,pf)
      return
```

```
10    if(yl(1).eq.0. .and. yl(2).eq.0.) go to 20
      call grv2hd(rho,yl,nvert,xb,zb,nfld,xf,zf,pf)
      return
20    call talwon(rho,nvert,xb,zb,nfld,xf,zf,pf)
      return
      end
c**********************************************************************
      subroutine talwon(rho,nvert,xb,zb,nobs,xf,zf,gc)
c  gravity profile using the Talwani 2-d prism algorithm
c  xb,zb,xf,zf in kilometers, rho in gm/cc
c  reference Talwani et al, JGR, v64, no1.
c  coded by Mike Webring 9/80
      dimension xb(1),zb(1),xf(1),zf(1),gc(1)
      if(rho.eq.0.0) return
      const=2.0*6.673
      pi=3.1415927
      hfpi=pi*.5
      twopi=2.0*pi
      tol=1.e-5
c
      do 200 j=1,nobs
      gtmp=0.0
c
      do 101 i=1,nvert
      capz=0.0
      il=i+1
      if(i.eq.nvert) il=1
      x1=xb(i)-xf(j)
      x2=xb(il)-xf(j)
      z1=zb(i)-zf(j)
      z2=zb(il)-zf(j)
      ax1=abs(x1)
      ax2=abs(x2)
      if(ax1.lt.tol .and. abs(z1).lt.tol) go to 100
      if(ax2.lt.tol .and. abs(z2).lt.tol) go to 100
      dx=x2-x1
      dz=z2-z1
      t1=atan2(z1,x1)
      t2=atan2(z2,x2)
      if(t1.lt.0.0) t1=t1+twopi
      if(t2.lt.0.0) t2=t2+twopi
      dt=t1-t2
      if(abs(dt).gt.pi) dt=dt-sign(twopi,dt)
      if(abs(dt).lt.tol) go to 100
c
      if(abs(dx).gt.tol) go to 10
      ct2=cos(t2)
      if(abs(ct2).lt.tol) go to 100
      f1=cos(t1)/ct2
      capz=x1*log(f1)
      go to 100
c
10    if(abs(dz).gt.tol) go to 20
      capz=-dt*z1
```

```
      go to 100
c
20    tphi=dz/dx
      dxs=dx*dx
      r2=dxs+dz*dz
      if(r2.lt.tol) go to 100
      acs=(dx*dz*x2-dxs*z2)/r2
      if(ax1.gt.tol) go to 30
      f2=sin(t2)-cos(t2)*tphi
      capz=-acs*(-dt+tphi*log(f2))
      go to 100
c
30    if(ax2.gt.tol) go to 40
      f1=sin(t1)-cos(t1)*tphi
      if(f1.le.0.0) then
      type *,j,xf(j),zf(j),xb(i),zb(i),xb(i1),zb(i1)
      stop
      endif
      capz=acs*(dt+tphi*log(f1))
      go to 100
c
40    r1s=x1*x1+z1*z1
      r2s=x2*x2+z2*z2
      capz=acs*(dt+tphi*0.5*log(r2s/r1s))
      go to 100
c
100   gtmp=gtmp+capz
101   continue
200   gc(j)=gc(j)+const*rho*gtmp
      return
      end
c**********************************************************************
      subroutine grv2hd(rho,yin,nvert,xb,zb,nobs,xf,zf,gc)
c  profile gravity calculation of a 2.5d prism
c  yin1 > yin2, z positive down
c  reference Rasmussen and Pedersen, Geophysical Prospecting, v27 p749-7
c  coded by Mike Webring, 11/81
      dimension x(2),y(2),yin(2),z(2),xzr(2),yr(2,2),f(4,2),ef(8)
      dimension xb(nvert),zb(nvert),xf(nobs),zf(nobs),gc(nobs)
      logical sym
      equivalence (f(1,1),ef(1))
c  rho in gm/cc, distance in kilometer
      const=6.673*rho
      tol=1.e-5
      pi=3.1415926
      twopi=2.*pi
      sym=.false.
      if(yin(1).eq.-yin(2)) sym=.true.
      it=2
      if(sym) it=1
      add=sign(1.,yin(1)*yin(2))
      y(1)=abs(yin(1))
      y(2)=abs(yin(2))
      if(y(1).gt.y(2)) go to 1
```

```
      y(1)=abs(yin(2))
      y(2)=abs(yin(1))
1     if(y(1).lt.tol .or. rho.eq.0.0) return
      if(y(2).lt.tol) it=1
c
      do 200 j=1,nobs
      x(1)=xb(1)-xf(j)
      z(1)=zb(1)-zf(j)
      r2=x(1)*x(1)+z(1)*z(1)
      xzr(1)=sqrt(r2)
      yr(1,1)=sqrt(r2+y(1)*y(1))
      yr(1,2)=sqrt(r2+y(2)*y(2))
c
      do 10 k=1,8
10    ef(k)=0.
      do 100 i=1,nvert
      i1=i+1
      if(i.eq.nvert) i1=1
      x(2)=xb(i1)-xf(j)
      z(2)=zb(i1)-zf(j)
      r2=x(2)*x(2)+z(2)*z(2)
      xzr(2)=sqrt(r2)
      dx=x(2)-x(1)
      dz=z(2)-z(1)
      dxz2=dx*dx+dz*dz
      dl=sqrt(dxz2)
      if(dl.lt.tol) go to 90
      cosp=dx/dl
      zn=-cosp
      sinp=dz/dl
      w=cosp*z(1)-sinp*x(1)
      u1=cosp*x(1)+sinp*z(1)
      u2=cosp*x(2)+sinp*z(2)
c
      do 80 ip=1,it
      yr(2,ip)=sqrt(r2+y(ip)*y(ip))
      f(1,ip)=f(1,ip)+zn*y(ip)*alog((u2+yr(2,ip))/(u1+yr(1,ip)))
      f(2,ip)=f(2,ip)+zn*u2*alog((y(ip)+yr(2,ip))/xzr(2))
      f(3,ip)=f(3,ip)+zn*u1*alog((y(ip)+yr(1,ip))/xzr(1))
      xn=u2*y(ip)
      xd=w*yr(2,ip)
      t2=atan2(xn,xd)
      if(t2.lt.0.0) t2=t2+twopi
      xn2=u1*y(ip)
      xd2=w*yr(1,ip)
      t1=atan2(xn2,xd2)
      if(t1.lt.0.0) t1=t1+twopi
      dt=t2-t1
      if(abs(dt).gt.pi) dt=dt-sign(twopi,dt)
      f(4,ip)=f(4,ip)+zn*w*dt
80    continue
90    xzr(1)=xzr(2)
      yr(1,1)=yr(2,1)
      yr(1,2)=yr(2,2)
```

```
      x(1)=x(2)
      z(1)=z(2)
100   continue
c
      sum1=f(1,1)-f(4,1)+f(2,1)-f(3,1)
      sum2=sum1
      if(.not.sym) sum2=f(1,2)-f(4,2)+f(2,2)-f(3,2)
      gf=sum1-add*sum2
      gc(j)=gc(j)-const*gf
200   continue
      return
      end
c*********************************************************************
      subroutine mag2hd(vm,tf,y,nvert,xb,zb,nobs,xf,zf,t)
c  computes the total field profile for a 2.5d prism
c  ref: shuey and pasquale, geophysics v38 n3
c  vm is the magnetization vector and tf is measurement unit vector
c  coded by mike webring, 10/81
      complex cdxz,cxz,ct1,ct2,cf,cfln
      dimension vm(3),tf(3),cf(2),x(2),z(2),r(2),
     1 xb(nvert),zb(nvert),xf(nobs),zf(nobs),t(nobs)
      if(y.gt.0.0) go to 1
      type *,' mag2hd: prism length <= zero, setting to 1000'
      y=1000.
1     ys=y*y
      do 50 iob=1,nobs
      px=0.
      pz=0.
      q=0.
      x(1)=xb(1)-xf(iob)
      z(1)=zb(1)-zf(iob)
      r(1)=sqrt(x(1)*x(1)+ys+z(1)*z(1))
c
      do 20 j=1,nvert
      j1=j+1
      if(j.eq.nvert) j1=1
      x(2)=xb(j1)-xf(iob)
      z(2)=zb(j1)-zf(iob)
        if(x(2).eq.0.0 .and. z(2).eq.0.0) then
        type *,' mag2hd: observation coincident with prism corner'
        return
      endif
      r(2)=sqrt(x(2)*x(2)+ys+z(2)*z(2))
      dx=x(2)-x(1)
      dz=z(2)-z(1)
      if(dx.eq.0.0 .and. dz.eq.0.0) then
        type *,' mag2hd: zero length segment, vertex ',j
        return
      endif
      cdxz=cmplx(dx,dz)
       do 10 k=1,2
       cxz=cmplx(x(k),z(k))
       t1=(1.0+r(k)/y)
       ct1=cmplx(t1,0.)
```

```
       t2=(x(k)*dz-z(k)*dx)/ys
       ct2=cmplx(0.,t2)
10     cf(k)=(cdxz*ct1)/cxz+ct2
      cfln=(clog(cf(2)/cf(1)))/cdxz
      px=px+dz*real(cfln)
      pz=pz-dx*aimag(cfln)
      q=q-dx*real(cfln)
      x(1)=x(2)
      z(1)=z(2)
20    r(1)=r(2)
c
      hx=2.0*(vm(1)*px+vm(3)*q)
      hy=2.0*vm(2)*(pz-px)
      hz=2.0*(vm(1)*q-vm(3)*pz)
50    t(iob)=t(iob)+hx*tf(1)+hy*tf(2)+hz*tf(3)
      return
      end
c*********************************************************************
      subroutine nlctl(ng,garr,nh,harr)
c  setup routine for simple parameters: x,z,rho,sus
c  watch place holding ng,nh vs. real ngrv,nmag dimensions
      common /parm1/  nvert,nbody,nobs1,mobs1,grvswt,magswt,
     1               iave(2),vex,istatn,iparsh,labelv,numbod
      common /model/  xb(100),zb(100),rho(50),sus(50)
      common /swork/  a(8000),err(400),s(20),u(8000),v(400)
      common /free1/  weight,rms1,iter,xk1,nfreex,nfreez,nfreer,nfrees,
     1               ifree(20),jfree(20),kfree(20),lfree(20)
      dimension garr(ng,5),harr(nh,5)
      logical grvswt,magswt
      data delta/1.0e-3/
c
c  get free parameter lists
      call getpar(npart,ier)
      if(ier.eq.1) return
c
c  get limits for active profile segment
100   call getact(igs,ngrv,ihs,nmag)
      no=ngrv+nmag
      if(no.le.2) return
c
c  get error vector
      do 101 i=1,ng
101   garr(i,4)=0.0
      do 102 i=1,nh
102   harr(i,4)=0.0
      iad=1
      if(grvswt)  then
                  call mardvr(0,xb,zb,rho,igs,ngrv,
     1            garr(igs,4),err(iad),rmsg)
                  iad=ngrv+1
      endif
      if(magswt)  call mardvr(1,xb,zb,sus,ihs,nmag,
     1            harr(ihs,4),err(iad),rmsh)
c
```

```
c  setup jacobian
       iad=1
       do 105 i=1,no*npart
105    a(i)=0.0
c
       if( nfreex.lt.1 ) go to 115
       dx=delta
       dz=0.0
       do 110 i=1,nfreex
       ivert=ifree(i)
       if(grvswt) call parshl(0,dx,dz,ivert,
      1          ngrv,garr(igs,1),garr(igs,2),a(iad))
       iad=iad+ngrv
       if(magswt) call parshl(1,dx,dz,ivert,
      1          nmag,harr(ihs,1),harr(ihs,2),a(iad))
110    iad=iad+nmag
c
115    if( nfreez.lt.1 ) go to 130
       dx=0.0
       dz=delta
       do 120 i=1,nfreez
       ivert=jfree(i)
       if(grvswt) call parshl(0,dx,dz,ivert,
      1          ngrv,garr(igs,1),garr(igs,2),a(iad))
       iad=iad+ngrv
       if(magswt) call parshl(1,dx,dz,ivert,
      1          nmag,harr(ihs,1),harr(ihs,2),a(iad))
120    iad=iad+nmag
c
130    dx=0.0
       dz=0.0
       if(nfreer.eq.0) go to 150
       do 140 i=1,nfreer
       iparm=kfree(i)
       if(grvswt) call parshl(0,dx,dz,iparm,
      1          ngrv,garr(igs,1),garr(igs,2),a(iad))
       iad=iad+ngrv
       if(magswt) then
       do 135 k=1,nmag
       a(iad)=0.0
135    iad=iad+1
       endif
140    continue
c
150    if(nfrees.eq.0) go to 170
       do 160 i=1,nfrees
       if(grvswt) then
       do 155 k=1,ngrv
       a(iad)=0.0
155    iad=iad+1
       endif
       iparm=lfree(i)
       call parshl(1,dx,dz,iparm,
      1    nmag,harr(ihs,1),harr(ihs,2),a(iad))
```

```
      iad=iad+nmag
160   continue
c
170   if(iparsh.eq.1) then
        type 171
171     format(' plot partial derivatives ? '$)
        if(noyes(1).eq.1) call advr(igs,ngrv,ihs,nmag,no,npart,a,err)
      endif
c
c  relative weighting of gravity to magnetics
      npxz=nfreex+nfreez
      call relsca(weight,ngrv,nmag,npxz,no,npart,a,err)
      if(iparsh.eq.1) then
        type 172
172     format(' plot weighted partial derivatives ? '$)
        if(noyes(1).eq.1) call advr(igs,ngrv,ihs,nmag,no,npart,a,err)
      endif
c
      if(iter.eq.1) then
      if(grvswt .and. magswt) then
        rms1=(weight*rmsg+rmsh)/(weight+1.0)
        type 175,rms1
175     format(1x,' weighted starting rms error',1pg15.5)
      else
        if(grvswt) rms1=rmsg
        if(magswt) rms1=rmsh
        type 176,rms1
176     format(' starting rms error',1pg15.5)
      endif
      endif
c
c  scale jacobian
      call cscale(no,npart,a,s)
      if(iparsh.eq.1) then
        type 173
173     format(' plot scaled system ? '$)
        if(noyes(1).eq.1) call advr(igs,ngrv,ihs,nmag,no,npart,a,err)
      endif
c
      if(iter.gt.1) go to 200
      type 180
180   format(' type parameter correlations ? '$)
      if(noyes(1).eq.1) then
        call crlmtx(6,no,npart,a)
        type 190
190     format(' continue with the inversion ? '$)
        if(noyes(1).eq.0) return
      endif
c
c  generate new parameters
200   call marq(no,npart,igs,ngrv,ihs,nmag)
      iter=iter+1
      type 210
210   format(' another iteration ? '$)
```

```
      if(noyes(1).eq.1) go to 100
999   return
      end
c**********************************************************************
      subroutine getact(igs,ngrv,ihs,nmag)
      common /field/  nf(10),f(2000)
      common /model/  xb(100),zb(100),rho(50),sus(50)
      common /parm1/  nvert,nbody,nobs,mobs,grvswt,magswt
      common /vertx/  icrn(500),ncrn(150),loca(150),nbord,ibord(100)
      common /free1/  weight,rms1,iter,xk1,nfreex,nfreez,nfreer,nfrees,
     1               ifree(20),jfree(20),kfree(20),lfree(20)
      logical grvswt,magswt
c  the partial derivatives should be approaching zero at pfact*depth.
      pfact=5.
      xmax=-1.e5
      xmin= 1.e5
      avedat= 1.e5
      if(grvswt) then
        aveg=0.0
        do 5 i=0,nobs-1
5       aveg=aveg+f(nf(2)+i)
        avedat=aveg/nobs
      endif
      if(magswt) then
        aveh=0.0
        do 10 i=0,mobs-1
10      aveh=aveh+f(nf(7)+i)
        aveh=aveh/mobs
        avedat=amin1(avedat,aveh)
      endif
c  free x's
      do ix=1,nfreex
      x=xb(ifree(ix))
      depth=abs(zb(ifree(ix))-avedat)
      x1=x-pfact*depth
      x2=x+pfact*depth
      if(x1.lt.xmin) xmin=x1
      if(x2.gt.xmax) xmax=x2
      end do
c  free z's
      do iz=1,nfreez
      x=xb(jfree(iz))
      depth=abs(zb(jfree(iz))-avedat)
      x1=x-pfact*depth
      x2=x+pfact*depth
      if(x1.lt.xmin) xmin=x1
      if(x2.gt.xmax) xmax=x2
      enddo
c  free rho and susceptibility
      do k=1,2
       nfree=0
       if(k.eq.1 .and. grvswt) nfree=nfreer
       if(k.eq.2 .and. magswt) nfree=nfrees
      do j=1,nfree
```

```
        ibody=kfree(j)
        if(k.eq.2) ibody=lfree(j)
       do i=1,ncrn(ibody)
       iadr=icrn(loca(ibody)+i)
       x=xb(iadr)
       depth=abs(zb(iadr)-avedat)
       x1=x-pfact*depth
       x2=x+pfact*depth
       if(x1.lt.xmin) xmin=x1
       if(x2.gt.xmax) xmax=x2
       enddo
       enddo
       enddo
c
c  appropriate indices
       igs=1
       ihs=1
       ngrv=0
       nmag=0
       if(grvswt) then
        ige=nobs
        x1=f(nf(1))
       do i=1,nobs-1
        x2=f(nf(1)+i)
        if(xmin.ge.x1 .and. xmin.lt.x2) igs=i
        if(xmax.gt.x1 .and. xmax.le.x2) ige=i+1
        x1=x2
       enddo
        ngrv=ige-igs+1
        xs = f(nf(1)+igs)
        xe = f(nf(1)+ige-1)
        type *,' active  gravity range (km)',xs,xe
       endif
c
       if(magswt) then
        ihe=mobs
        x1=f(nf(6))
       do i=1,mobs-1
        x2=f(nf(6)+i)
        if(xmin.ge.x1 .and. xmin.lt.x2) ihs=i
        if(xmax.gt.x1 .and. xmax.le.x2) ihe=i+1
        x1=x2
       enddo
        nmag=ihe-ihs+1
        xs = f(nf(6)+ihs)
        xe = f(nf(6)+ihe-1)
        type *,' active magnetic range (km)',xs,xe
       endif
       return
       end
c**********************************************************************
       subroutine getpar(npart,ier)
c  enter the free parameters x,z,rho,sus
       common /parm1/  nvert,nbody,nobs,mobs,grvswt,magswt
```

```
      common /free1/  weight,rms1,iter,xkl,nfreex,nfreez,nfreer,nfrees,
     1               ifree(20),jfree(20),kfree(20),lfree(20)
      logical grvswt,magswt
      data itty/5/
c
      ier=0
5     npart=nfreex+nfreez+nfreer+nfrees
      if(npart.ne.0) then
        type 6
6       format(' reset free parameters ? '$)
        if(noyes(1).eq.0) go to 200
      endif
      npart=0
      nfreer=0
      nfrees=0
      iter=1
9     type  10
10    format(' enter number of free x vertices:'$)
      read(itty,*,end=999,err=999) nfreex
      if(nfreex.lt.0) go to 101
      ntmp=nfreex
      if(nfreex.gt.nvert .or. ntmp.gt.20) go to 120
      if(nfreex.eq.0) go to 15
      type  24
      read(itty,*,end=9,err=9) (ifree(i),i=1,nfreex)
      do 12 i=1,nfreex
      if(ifree(i).lt.1 .or. ifree(i).gt.nvert) go to 9
12    continue
c
15    type  20
20    format(' enter number of free z vertices:'$)
      read(itty,*,end=999,err=999) nfreez
      if(nfreez.lt.0) go to 101
      ntmp=nfreez+nfreex
      if(nfreez.gt.nvert .or. ntmp.gt.20) go to 120
      if(nfreez.eq.0) go to 25
      type  24
24    format(' enter indices : '$)
      read(itty,*,end=15,err=15) (jfree(i),i=1,nfreez)
      do 22 i=1,nfreez
      if(jfree(i).lt.1 .or. jfree(i).gt.nvert) go to 15
22    continue
c
25    if(.not.grvswt) go to 70
      type  50
50    format(' enter number of free densities :'$)
      read(itty,*,end=999,err=999) nfreer
      if(nfreer.lt.0) go to 101
      ntmp=nfreer+nfreez+nfreex
      if(nfreer.gt.nbody .or. ntmp.gt.20) go to 120
      if(nfreer.eq.0) go to 70
      type  24
      read(itty,*,end=25,err=25) (kfree(i),i=1,nfreer)
      do 60 i=1,nfreer
```

```
      if(kfree(i).lt.1 .or. kfree(i).gt.nbody) go to 25
60    continue
c
70    if(.not.magswt) go to 100
      type  80
80    format(' enter number of free susceptibilites :'$)
      read(itty,*,end=999,err=999) nfrees
      if(nfrees.lt.0) go to 101
      ntmp=nfrees+nfreer+nfreez+nfreex
      if(nfrees.gt.nbody .or. ntmp.gt.20) go to 120
      if(nfrees.eq.0) go to 100
      type  24
      read(itty,*,end=70,err=70) (lfree(i),i=1,nfrees)
      do 90 i=1,nfrees
      if(lfree(i).lt.1 .or. lfree(i).gt.nbody) go to 70
90    continue
c
c  npartials check
100   npart=nfreex+nfreez+nfreer+nfrees
101   if(npart.gt.0) go to 110
      type *,' none set'
      nfreex=0
      nfreez=0
      nfreer=0
      nfrees=0
      go to 999
110   if(npart.le.20) go to 200
      type *,' maximum number free is 20'
      go to 999
120   type *,' exceeded either number of vertices or bodies'
      type *,' or a total of 20 parameters free.'
      go to 999
c
200   return
999   ier=1
      return
      end
c**********************************************************************
      subroutine parshl(mgswt,dx,dz,iparm,nfld,xf,zf,pf)
c  partial derivatives for x,z,rho,sus via central differences
      common /parm1/  nvert,nbody
      common /model/  xb(100),zb(100),rho(50),sus(50),pl(50,2),rem(50,3)
      common /vertx/  icrn(500),ncrn(150),loca(150)
      common /magxyz/ ef(3),ev(3),tdir(3),azmuth
      dimension vm(3),rv(3),yl(2),xbl(100),zbl(100),tmp(400)
      dimension xf(nfld),zf(nfld),pf(nfld)
      data d2r/1.745329e-2/
      do 1 i=1,nfld
      tmp(i)=0.0
1     pf(i)=0.0
c
      if(dx.eq.0.0 .and. dz.eq.0.0) go to 200
      do 100 jbody=1,nbody
      is=1
```

```
      ie=ncrn(jbody)
      if(iparm.le.0) go to 21
      do 10 ip=1,ncrn(jbody)
      if(icrn(loca(jbody)+ip).eq.iparm) go to 20
10    continue
      go to 100
20    is=ip
      ie=ip
21    if(mgswt.eq.0) then
        alin=rho(jbody)
      else
        alin=sus(jbody)
      endif
      do 30 i=1,ncrn(jbody)
      xbl(i)=xb(icrn(loca(jbody)+i))
30    zbl(i)=zb(icrn(loca(jbody)+i))
      do 40 i=is,ie
      xbl(i)=xbl(i)+dx
40    zbl(i)=zbl(i)+dz
      jl=jbody
      call pickmg(mgswt,jl,alin,ncrn(jbody),xbl,zbl,
     1 nfld,xf,zf,pf)
      do 50 i=is,ie
      xbl(i)=xbl(i)-2.0*dx
50    zbl(i)=zbl(i)-2.0*dz
      call pickmg(mgswt,jl,alin,ncrn(jbody),xbl,zbl,
     1 nfld,xf,zf,tmp)
100   continue
      delta=dx
      if(dx.eq.0.0) delta=dz
      delta=1.0/(2.0*abs(delta))
      go to 300
c
200   do i=1,ncrn(iparm)
        xbl(i)=xb(icrn(loca(iparm)+i))
        zbl(i)=zb(icrn(loca(iparm)+i))
      enddo
      if(mgswt.eq.0) then
        alin=1.0
        call pickmg(0,iparm,alin,ncrn(iparm),xbl,zbl,nfld,xf,zf,pf)
        return
      else
        h=abs(sus(iparm)*ef(1)) + abs(rem(iparm,1)*1.0e5)
        ds=h*1.0e-5
        if(ds.lt.1.e-10) ds=1.e-10
        alin=sus(iparm)+ds
        call pickmg(1,iparm,alin,ncrn(iparm),xbl,zbl,nfld,xf,zf,pf)
        alin=sus(iparm)-ds
        call pickmg(1,iparm,alin,ncrn(iparm),xbl,zbl,nfld,xf,zf,tmp)
        delta=1.0/(2.0*ds)
      endif
c
300   do i=1,nfld
      pf(i)=delta*(pf(i)-tmp(i))
```

```
      enddo
      return
      end
c*********************************************************************
      subroutine crlmtx(jdev,m,n,a)
c  print the triangular correlation matrix
      dimension icl(20),a(m,n)
      write(jdev,5)
5     format(1x,' a=')
      do 20 j=2,n
      do 10 i=1,j-1
10    icl(i)=int( 100.0*correl(m,a(1,j),a(1,i)) + 0.5)
      write(jdev,15) j,(icl(i),i=1,j-1)
15    format(1x,i3,20i3)
20    continue
      write(jdev,30)
30    format(/,1x,' b=  1  2  3  4  5  6  7  8  9 10 11 12 13 14 1
      return
      end
c*********************************************************************
      subroutine relsca(weight,ngrv,nmag,npxz,no,np,a,err)
c  row scaling for relative mag/grv weight
      common /parm1/ fill(4),grvswt,magswt
      dimension a(no,np),err(no)
      logical grvswt,magswt
      weight=1.0
      if(.not.(grvswt.and.magswt)) return
      call ammi(ngrv,err,        gmin,gmax,mn,mx,1)
      call ammi(nmag,err(ngrv+1),hmin,hmax,mn,mx,1)
      gmax=amax1(abs(gmin),abs(gmax))
      hmax=amax1(abs(hmin),abs(hmax))
      ratio=1.0
      if(gmax*hmax.gt.1.e-10) ratio=gmax/hmax
      do j=1,npxz
        call ammi(ngrv,a(1,j),     gmin,gmax,mn,mx,1)
        call ammi(nmag,a(ngrv+1,j),hmin,hmax,mn,mx,1)
        gmax=amax1(abs(gmin),abs(gmax))
        hmax=amax1(abs(hmin),abs(hmax))
        r=1.0
        if(gmax*hmax.gt.1.e-10) r=gmax/hmax
        ratio=ratio+r
      enddo
      ratio=ratio/float(npxz+1)
c
      relwt=weight*ratio
      do 50 j=1,np
      do 50 i=ngrv+1,no
50    a(i,j)=relwt*a(i,j)
      do 100 i=ngrv+1,no
100   err(i)=relwt*err(i)
      return
      end
c*********************************************************************
      subroutine cscale(m,n,a,s)
```

```
c  column scaling for matrix a
      dimension s(n),a(1)
      tol=1.e-10
      do 10 j=1,n
      k=(j-1)*m+1
      pmin=a(k)
      pmax=a(k)
      do 5 i=1,m
      if(a(k).gt.pmax) pmax=a(k)
      if(a(k).lt.pmin) pmin=a(k)
5     k=k+1
      s(j)=amax1(abs(pmax),abs(pmin))
      if(s(j).lt.tol) s(j)=1.0
10    s(j)=1.0/s(j)
      k=1
      do 20 j=1,n
      do 20 i=1,m
      a(k)=a(k)*s(j)
20    k=k+1
      return
      end
c*********************************************************************
      subroutine marq(no,np,igs,ngrv,ihs,nmag)
      common /field/  nf(10),f(2000)
      common /parm1/  nv,nb,ifill(2),grvswt,magswt
      common /model/  xb(100),zb(100),rho(50),sus(50)
      common /free1/  weight,rms1,iter,xk1,nfx,nfz,nfr,nfs,
     1               ifx(20),ifz(20),ifr(20),ifs(20)
      common /swork/  a(8000),err(400),s(20),u(8000),v(400)
c
      dimension q(20),  work(80),ute(20),qute(20),dp(20),dg(20)
      equivalence (work,ute),    (work(21),qute),
     1           (work(41),dp), (work(61),dg)
      dimension xb1(100),zb1(100),rho1(50),sus1(50)
      dimension nf1(10),xk(3),rms(3),ang(3),dp2(3)
c
      character cp*1
      logical grvswt,magswt
      data attn/4./,mxridg/10/
c  eigenvalue attenuation rate set by attn
c
      do i=1,5
        nf1(i)  = nf(i)  +(igs-1)
        nf1(i+5)= nf(i+5)+(ihs-1)
      enddo
      icount=0
      istop=0
      limit=0
      init=1
      ii=3
c
      call svd(a,no,np,q,u,v,work)
      if(q(1).eq.0.0) then
        type *,' no independent parameters'
```

```
        return
      endif
      type *,' scaled eigenvalues'
      do i=1,np
        ute(i)=q(i)/q(1)
      enddo
      type 10,(ute(i),i=1,np)
10    format(4(1x,5f8.4,/))
c
      call atb(u,no,np,err,no,1,ute)
      call ata(u,no,np,a)
      call chki(a,np,icode)
      call ata(v,np,np,a)
      call chki(a,np,jcode)
      if(icode.lt.6) type *,'svd: UtU',icode
      if(jcode.lt.6) type *,'svd: VtV',jcode
c
      call ammi(np,q,qmin,qmax,mn,mx,1)
      xk(3) = 1.5*qmax
      if (iter.gt.1)  xk(3) = 1.5*xk1
c
c  steepest descent vector
      damp=10.*qmax**attn
      do 50 i=1,np
50    qute(i)=ute(i)/(q(i)+damp)
      call axb(v,np,np,qute,np,1,dg)
      dgl=0.
      do 80 i=1,np
      dg(i)=dg(i)*s(i)
80    dgl=dgl+dg(i)*dg(i)
      dgl=sqrt(dgl)
c
c  system damping loop
      type  90
90    format('  cutoff     rms      angle     delta p')
100   icount=icount+1
      if((limit.eq.1  .and. init.eq.0) .or. icount.gt.mxridg) then
        istop=1
        call ammi(3,rms,emin,emax,ii,mx,1)
        call ammi(3,ang,amin,amax,iia,mx,1)
        if(iia.gt.ii) ii=iia
        if (rms(ii).gt.rms1) ii=3
        thres=xk(ii)/q(1)
        type 101,thres
101     format(' choosing eigenvalue cutoff',f8.4)
      endif
c
c  get new correction vector
      damp=xk(ii)**attn
      do 104 i=1,np
      q2=q(i)**(attn-1.)
104   qute(i)=ute(i)*(q2/(q2*q(i)+damp))
      call axb(v,np,np,qute,np,1,dp)
      do 105 i=1,np
```

```
105   dp(i)=dp(i)*s(i)
c
c  assemble and check parameter vector
      call corvec(limit,np,dp,xbl,zbl,rhol,susl,istop)
c
c  forward calculation
200   if(grvswt) call mardvr(0,xbl,zbl,rhol,igs,ngrv,
     1          f(nfl(4)),f(nfl(5)),rmsg)
      if(magswt) call mardvr(1,xbl,zbl,susl,ihs,nmag,
     1          f(nfl(9)),f(nfl(10)),rmsm)
      rms(ii)=rmsg
      if(magswt) rms(ii)=rmsm
      if(grvswt .and. magswt) rms(ii)=(weight*rmsg+rmsm)/(weight+1.0)
      dp2(ii)=0.0
      an=0.0
      dpl=0.0
      do 210 i=1,np
        if(i.le.nfx+nfz) dp2(ii) = dp2(ii) + dp(i)*dp(i)
        dpl = dpl + dp(i)*dp(i)
210     an  = an  + dp(i)*dg(i)
      dpl = sqrt(dpl)
      if(dpl.lt.1.e-10) go to 300
      ad = dpl*dgl
      ang(ii) = acosd2(an,ad)
      if(nfx+nfz.gt.0) dp2(ii) = sqrt( dp2(ii)/float(nfx+nfz) )
      thres = xk(ii)/q(1)
      type  220, thres, rms(ii), ang(ii), dp2(ii)
220   format(1x,f8.4,g12.4,f7.2,f10.3)
      if(istop.eq.1) go to 300
c
c  iteration control
      if(init.eq.1 .and. ii.gt.1) then
        ii=ii-1
        xk(ii)=xk(ii+1)/2.
      else
        init=0
        call bictl(xk,rms,ang,dp2,ii,ireset)
        if(ireset.eq.-1) limit=1
      endif
      go to 100
c
c  print new parameter values
300   type  310
310   format(' parm   change       new value')
      do 390 i=1,np
      if(i.gt.nfx) go to 320
      cp='x'
      j=ifx(i)
      p=xbl(j)
      go to 380
320   if(i.gt.(nfx+nfz)) go to 330
      cp='z'
      j=ifz(i-nfx)
      p=zbl(j)
```

```
      go to 380
330   if(i.gt.(nfx+nfz+nfr)) go to 340
      cp='r'
      j=ifr(i-(nfx+nfz))
      p=rhol(j)
      go to 380
340   cp='s'
      j=ifs(i-(nfx+nfz+nfr))
      p=susl(j)
380   type  381,cp,j,dp(i),p
381   format(1x,a1,i2,5x,1p2(g12.4,1x))
390   continue
c
c  save new parameter values
      rmsimp=0.0
      if(rms1.gt.0.0) rmsimp=100.*(rms1-rms(ii))/rms1
      type  410,iter,rms(ii),rmsimp
410   format(' iteration ',i2,',  rms error is  ',1pg15.4,/,
     1 ' percent improvement is  ',1pg15.4)
      type  420
420   format(' save new parameter values ?'$)
      if(noyes(1).eq.0) return
      do 430 i=1,nv
      xb(i)=xbl(i)
430   zb(i)=zbl(i)
      do 440 i=1,nb
      if(magswt) sus(i)=susl(i)
      if(grvswt) rho(i)=rhol(i)
440   continue
      xk1=xk(ii)
      rms1=rms(ii)
      return
      end
c**********************************************************************
      subroutine corvec(limit,np,dp,xbl,zbl,rhol,susl,iprnt)
      common /field/  nf(10),f(2000)
      common /parm1/  nv,nb,ifill(2),grvswt,magswt
      common /model/  xb(100),zb(100),rho(50),sus(50)
      common /free1/  fill(4),nfx,nfz,nfr,nfs,
     1               ifx(20),ifz(20),ifr(20),ifs(20)
c
      dimension dp(20),xbl(100),zbl(100),rhol(50),susl(50)
      data rlimit/.05/, xfrac,zfrac/50.0,10.0/
      logical grvswt,magswt,limitp
c
      limit=0
c  get x&z excursion limits
      x1=0.0
      x2=0.0
      if(grvswt) x1 = abs(f(nf(2)-1)-f(nf(1)) )
      if(magswt) x2 = abs(f(nf(7)-1)-f(nf(6)) )
      xlimit = amax1(x1,x2) / xfrac
      call ammi(nv,zb,zmn,zmx,mn,mx,1)
      zlimit = (zmx-zmn) / zfrac
```

```
c
c  assemble parameter vector
      do 10 i=1,nv
      xbl(i)=xb(i)
10    zbl(i)=zb(i)
      do 20 i=1,nb
      susl(i)=sus(i)
20    rhol(i)=rho(i)
c
c  add in correction vector and check excursion
      ip=0
c  x parameter
      do i=1,nfx
      limitx=0
      ip=ip+1
      j=ifx(ip)
50      absdp = abs(dp(ip))
        xbl(j)= xb(j)+dp(ip)
        if(absdp.gt.xlimit .or. limitp(j,nb,xbl,zbl,0)) then
          dp(ip)=dp(ip)/2.0
          if(absdp.lt.0.01) dp(ip)=0.0
          limitx=j
          dpx=dp(ip)
        if(dpx.ne.0.0) go to 50
      endif
      if(limitx.gt.0) then
        if(iprnt.eq.1)
     1   type *,' limited x excursion of parameter',limitx,' to',dpx
        limit=1
        endif
      enddo
c  z parameter
      do i=1,nfz
      limitz=0
      ip=ip+1
      j=ifz(i)
100     absdp=abs(dp(ip))
        zbl(j)=zb(j)+dp(ip)
        if(absdp.gt.zlimit .or. limitp(j,nb,xbl,zbl,0)) then
          dp(ip)=dp(ip)/2.0
          if(absdp.lt.0.01) dp(ip)=0.0
          limitz=j
          dpz=dp(ip)
      if(dp(ip).ne.0.0) go to 100
      endif
      if(limitz.gt.0) then
        if(iprnt.eq.1)
     1  type *,' limited z excursion of parameter',limitz,' to',dpz
        limit=1
        endif
      enddo
c  rho
      do i=1,nfr
        ip=ip+1
```

```
        j=ifr(i)
        if(abs(dp(ip)).gt.rlimit) dp(ip)=sign(rlimit,dp(ip))
        rho1(j)=rho(j)+dp(ip)
      enddo
c  susceptibility
      do i=1,nfs
        ip=ip+1
        j=ifs(i)
        sus1(j)=sus(j)+dp(ip)
      enddo
c
      return
      end
c*********************************************************************
      subroutine mardvr(mgswt,xb,zb,alin,istart,np,pf,pe,rms)
c  forward driver for the inversion section where xb,zb,alin are
c  updated parameter vectors.
      common /field/ nf(10),f(2000)
      common /parm1/ nvert,nbody,nobs,mobs,grvswt,magswt,iave(2)
      common /swork/ a(16600),xb1(100),zb1(100)
      common /vertx/ icrn(500),ncrn(150),loca(150),nbord,ibord(100)
      dimension xb(100),zb(100),alin(50)
      dimension pf(np),pe(np)
      logical inside,grvswt,magswt
c  initialize output
      rms=1.e30
      do 10 i=1,np
10    pf(i)=0.0
c  find observed data
      noff=0
      if(mgswt.eq.1) noff=5
      ix = nf(1+noff) + (istart-1)
      iz = nf(2+noff) + (istart-1)
      iobs=nf(3+noff) + (istart-1)
c
c  call forward selection routine
      do 100 j=1,nbody
      do 50 i=1,ncrn(j)
      iadr=icrn(loca(j)+i)
      xb1(i)=xb(iadr)
50    zb1(i)=zb(iadr)
      jbody=j
      call pickmg(mgswt,jbody,alin(j),ncrn(j),xb1,zb1,
     1  np,f(ix),f(iz),pf)
100   continue
c
c  float datum
      if(iave(mgswt+1).eq.0) go to 150
      ave=0.0
      i1=iobs
      do 110 i=1,np
      ave=ave+f(i1)-pf(i)
110   i1=i1+1
      ave=ave/float(np)
```

```
      do 120 i=1,np
120   pf(i)=pf(i)+ave
c
c  error vector
150   errt=0.0
      il=iobs
      do 160 i=1,np
      pe(i)=f(il)-pf(i)
      errt=errt+pe(i)*pe(i)
160   il=il+1
      rms=sqrt(errt/float(np))
      return
      end
c**********************************************************************
      subroutine bictl(x,err,ang,dp2,ii,ireset)
c  locate the next eigenvalue cutoff by checking the rms error and the
c  angle formed between test vectors and the steepest descent vector
c  and choose the more conservative minimum.
c  ireset=-1 stop,             ireset=0 interval division,
c  ireset=1 interval addition, ireset=2 balance intervals
      dimension x(3),err(3),ang(3),dp2(3)
      ireset=-1
      if(x(3).le.x(2) .or. x(2).le.x(1)) return
c  check for convergence
      if(err(2).eq.0.0) return
      derr = ( abs(err(3)-err(2)) + abs(err(2)-err(1)) )/ err(2)
      if(derr.lt.0.001) return
c  balance intervals
      ratio=(x(3)-x(2))/(x(2)-x(1))
      if(ratio.gt.5.0 .or. ratio.lt.0.2) then
        ireset=2
        ii=2
        x(2)=0.5*(x(3)+x(1))
        go to 100
      endif
c
      icase=0
      se2=(err(2)-err(1))/(x(2)-x(1))
      se3=(err(3)-err(2))/(x(3)-x(2))
      if(se2.gt.0. .and. se3.gt.0.0) icase=1
      if(se2.lt.0. .and. se3.lt.0.0) icase=4
      if(icase.eq.0) then
        icase=2
        if(abs(se2).gt.se3) icase=3
      endif
c
      jcase=0
      sa2=(ang(2)-ang(1))/(x(2)-x(1))
      sa3=(ang(3)-ang(2))/(x(3)-x(2))
      if(sa2.gt.0. .and. sa3.gt.0.0) jcase=1
      if(sa2.lt.0. .and. sa3.lt.0.0) jcase=4
      if(jcase.eq.0) then
        jcase=2
        if(abs(sa2).gt.sa3) jcase=3
```

```
      endif
c
c  override an interval division by a good angle improvement
      if (icase.ne.4) then
        dangl = abs(ang(3)-ang(2)) + abs(ang(2)-ang(1))
        if (icase.le.2 .and. (jcase.gt.2 .and. dangl.gt.5.0) )
     1  icase=(icase+jcase)/2
      endif
c
      ireset=1
      if(icase.eq.1) then
        x(3)=x(2)
        x(2)=x(1)
        x(1)=x(2)/2.
        err(3)=err(2)
        err(2)=err(1)
        ang(3)=ang(2)
        ang(2)=ang(1)
        dp2(3)=dp2(2)
        dp2(2)=dp2(1)
      else if(icase.eq.2 .or. icase.eq.3) then
        if(icase.eq.2) x(1)=(x(1)+x(2))/2.
        if(icase.eq.3) x(3)=(x(2)+x(3))/2.
        ireset=0
      else if(icase.eq.4) then
        x(1)=x(2)
        x(2)=x(3)
        x(3)=1.5*(x(2)-x(1))+x(2)
        err(1)=err(2)
        err(2)=err(3)
        ang(1)=ang(2)
        ang(2)=ang(3)
        dp2(1)=dp2(2)
        dp2(2)=dp2(3)
      endif
      ii=1
      if(icase.gt.2) ii=3
100   err(ii)=1.0e30
      ang(ii)=90.0
      return
      end
c**********************************************************************
      logical function limitp(m,nbody,xb,zb,iprnt)
      common /vertx/ lookup(500),ncrn(150),loca(150)
      dimension lint(2),xb(1),zb(1)
      logical planar
      limitp=.false.
      do 10 j=1,nbody
        do 10 i=1,ncrn(j)
        k=lookup(loca(j)+i)
        if(k.eq.m) then
          ic=loca(j)+1
          limitp=.not.planar(ncrn(j),xb,zb,lookup(ic),lint)
          if(iprnt.ne.0) then
```

```
            type 20, j,lint
          endif
          if(limitp) return
        endif
10    continue
20    format(' body',i3,' has intersecting line segments',i3,'and'
      return
      end
c*********************************************************************
      logical function planar(n,x,z,lookup,lint)
c  check polygon for intersecting line segments
      dimension lint(2),lookup(n),x(1),z(1)
      lint(1)=0
      lint(2)=0
      planar=.true.
      if(n.eq.3) return
      tolfac=.0001
c  first line segment
      do 10 line1=1,n-2
      i2=line1+1
      if(i2.gt.n) i2=1
      i=lookup(line1)
      i2=lookup(i2)
      dz=z(i2)-z(i)
      dx=x(i2)-x(i)
      adx=abs(dx)
      tol=tolfac*(adx+abs(dz)) + 1.e-10
      if(adx.lt.tol) dx=tol
      am=dz/dx
      amn=amin1(x(i),x(i2))-tol
      amx=amax1(x(i),x(i2))+tol
      aymn=amin1(z(i),z(i2))-tol
      aymx=amax1(z(i),z(i2))+tol
      k=n
      if(line1.eq.1) k=n-1
c  compared to segments: +2 to n segment-1
      do 10 line2=line1+2,k
      j2=line2+1
      if(j2.gt.n) j2=1
      j=lookup(line2)
      j2=lookup(j2)
      dz=z(j2)-z(j)
      dx=x(j2)-x(j)
      adx=abs(dx)
      tol=tolfac*(adx+abs(dz)) + 1.e-10
      if(adx.lt.tol) dx=tol
      bm=dz/dx
      dslope=am-bm
      if(abs(dslope).lt.tolfac) go to 10
      rs=1.0/dslope
      xi=(rs*am)*x(i)-(rs*bm)*x(j)+rs*(z(j)-z(i))
      yi=am*xi-am*x(i)+z(i)
      bmn=amin1(x(j),x(j2))-tol
      bmx=amax1(x(j),x(j2))+tol
```

```
      bymn=amin1(z(j),z(j2))-tol
      bymx=amax1(z(j),z(j2))+tol
c  test for intersection within segments
      if(xi.lt.amn .or. xi.lt.bmn) go to 10
      if(xi.gt.amx .or. xi.gt.bmx) go to 10
      if(yi.lt.aymn .or. yi.lt.bymn) go to 10
      if(yi.gt.aymx .or. yi.gt.bymx) go to 10
c  intersection between line segments
      lint(1)=line1
      lint(2)=line2
      planar=.false.
      go to 30
10    continue
30    return
      end
c*********************************************************************
      subroutine setup
c  runtime modification of control parameters
      common /parm1/ nvert,nbody,nobs,mobs,grvswt,magswt,
     1 iave,jave,vex,istatn,iparsh,labelv,numbod
      common /plot1/  iplotr,sizea,sizet,sizel,
     1               it1(14),it2(14),it3(14),
     1               xscale,xxx(2),adelx,lintx,itx(5),
     1               zscale,zzz(2),adelz,lintz,itz(5),
     1               gscale,ggg(2),adelg,lintg,itg(5),
     1               hscale,hhh(2),adelh,linth,ith(5)
      common /magxyz/ ef(3),ev(3),tdir(3),azmuth,icompn
      common /pltdev/ jplotr,xboard,yboard,xlimit,ylimit
      character ians*16,ch*1,swt*3
      logical grvswt,magswt
      itty=5
      ierr=0
c
10    type 11
11    format(' setup parameters :'$)
      read(itty,37,end=999) ians
37    format(a16)
      call cvc(ians,2)
      n=leftj(ians)
      ch=ians(1:1)
      if(.not.(ch.eq.'d' .or. ch.eq.'m' .or. ch.eq.'p' .or. ch.eq.'s'
     1  .or. ch.eq.'r')) then
      type *,' setup parameters are :'
      type *,' datum type, magnetic direction'
      type *,' plot parameters, summary, or return'
      if(ierr.ge.2) return
      ierr=ierr+1
      go to 10
      endif
      ierr=0
c
c  datum type
      if(ians(1:1).eq.'d') then
      if(grvswt) then
```

```
        type 20
20      format(' set floating gravity datum ? '$)
        iave=noyes(1)
      endif
      if(magswt) then
        type 25
25      format(' set floating magnetic datum ? '$)
        jave=noyes(1)
      endif
      go to 10
      endif
c
c  magnetic parameters
      if(ians(1:1).eq.'m') then
      call initm(1)
      go to 10
      endif
c
c  plot parameters
      if(ians(1:1).eq.'p') then
        type *,' current plot device =',jplotr
        type *,' do you want to change device ?'
        if(noyes(1dum).eq.1) then
          type *,' enter new device #: 0=Calcomp,1=Tek,5=HP,6=Vers'
          read(5,*,end=10,err=10) iplotr
          if(iplotr.ne.jplotr) then
            xlimit=999.
            ylimit=999.
            jplotr=iplotr
          endif
          if(jplotr.ne.1) then
            type *,' enter 0 for an overall plot size '
            type *,' or    1 for separate scaling factors '
            read(5,*,end=10,err=10) ifunc
            if(ifunc.eq.0) then
              type *,' enter maximum x,y plot size in inches '
              read(5,*,end=10,err=10) xlimit,ylimit
            else
              type *,' enter x,g,h scaling factors'
              read(5,*,end=10,err=10) xscale,gscale,hscale
            endif
          endif
        endif
        if(iplotr.eq.1) then
          xscale=0.0
          zscale=0.0
          gscale=0.0
          hscale=0.0
        endif
      type *,' (re)set vertical exaggeration and windows ?'
      if(noyes(1dum).eq.1) then
        type *,' current vertical exaggeration =',vex
        type *,' enter new vertical exaggeration for the model plot'
        read(5,*,end=10,err=10) vex
```

```
        type *,' enter 0 0 to allow windows to default'
        type *,' model vertical window (min,max) :'
        read(5,*,end=10,err=10) zzz(1),zzz(2)
        if (grvswt) type *,' gravity window (min,max)  :'
        if (grvswt) read(5,*,end=10,err=10) ggg(1),ggg(2)
        if (magswt) type *,' magnetic window (min,max) :'
        if (magswt) read(5,*,end=10,err=10) hhh(1),hhh(2)
      endif
      type *,' continue with secondary switches ?'
      if(noyes(1dum).eq.0) go to 10
        type 32
32      format(' plot station locations ? '$)
        istatn=noyes(1)
        type 33
33      format(' plot body numbers ? '$)
        numbod=noyes(1)
        type 34
34      format(' plot vertex numbers ? '$)
        labelv=noyes(1)
        type 35
35      format(' enable partial derivative plots'/,
     1 ' in the inversion function ? '$)
        iparsh=noyes(1)
      go to 10
39    type *,' input error'
      go to 10
      endif
c
      if(ians(1:1).eq.'s') then
      type *
      type *,' datum types are'
      if(grvswt) then
      if(iave.eq.1) then
        type *,' floating gravity'
      else
        type *,' nonfloating gravity'
      endif
      endif
      if(magswt) then
      if(jave.eq.1) then
        type *,' floating magnetics'
      else
        type *,' nonfloating magnetics'
      endif
      type *
      type *,' magnetic field parameters are'
      type 40,ef
40    format(' intensity=',1pe15.8,/,' inclination=',0pf7.2,/,
     1 ' declination=',f7.2)
      type *,' profile azimuth=',azmuth
      if(icompn.eq.0) then
        type *,' total field is calculated'
      else
        dir='t'
```

```
        if(icompn.eq.1) dir='x'
        if(icompn.eq.2) dir='y'
        if(icompn.eq.3) dir='z'
        type 42,dir
42      format(1x,a1,' component is calculated')
      endif
      endif
      type *
      type *,'plot parameters are'
      type *,' vertical exaggeration =',vex
      swt='off'
      if(istatn.eq.1) swt='on'
      type *,' station location ',swt
      swt='off'
      if(numbod.eq.1) swt='on'
      type *,' body numbers ',swt
      swt='off'
      if(labelv.eq.1) swt='on'
      type *,' vertice labels ',swt
      swt='off'
      if(iparsh.eq.1) swt='on'
      type *,' partial derivative plots ',swt
      go to 10
      endif
c
      if(ians(1:1).eq.'r') return
      go to 10
999   return
      end
c**********************************************************************
      subroutine initw(xxx,grvswt,magswt)
      common /field/  nf(10),f(2000)
      dimension xxx(2)
      logical grvswt,magswt
      xxx(1)=1.e10
      xxx(2)=-xxx(1)
      if(grvswt) then
        xxx(1)=f(nf(1))
        xxx(2)=f(nf(2)-1)
      endif
      if(magswt) then
        if(f(nf(6)  ).lt.xxx(1)) xxx(1)=f(nf(6))
        if(f(nf(7)-1).lt.xxx(2)) xxx(2)=f(nf(7)-1)
      endif
      return
      end
c**********************************************************************
      subroutine initnf(nobs,mobs,nf,max,ierr)
      dimension nf(10)
      ierr=0
      if(5*(nobs+mobs).gt.max) go to 3
      nf(1)=1
      do 1 i=2,6
1     nf(i)=nf(i-1)+nobs
```

```
      do 2 i=7,10
2     nf(i)=nf(i-1)+mobs
      return
3     type 4,max
4     format(' data requires more than the ',i4,' words allocated')
      ierr=1
      return
      end
c*********************************************************************
      subroutine initl(plenth,pl,nbody,ireset)
      dimension pl(2,50)
      if(plenth.lt.0.0) go to 15
      do 10 j=1,nbody
      if(ireset.eq.1) go to 5
      if(plenth.eq.0.0) go to 10
      if(pl(1,j).ne.0. .or. pl(2,j).ne.0.) go to 10
5     pl(1,j)=plenth
      pl(2,j)=-plenth
10    continue
      return
15    type 20
20    format(' initl: no change, plenth must be >= 0.')
      return
      end
c*********************************************************************
      subroutine initp
      common /pltdev/ jplotr,xboard,yboard,xlimit,ylimit
      common /plot1/ iplotr,sizea,sizet,sizel,
     1 title1,title2,title3,
     1 xscale,xxx(2),adelx,lintx,titlx,
     1 zscale,zzz(2),adelz,lintz,titlz,
     1 gscale,ggg(2),adelg,lintg,titlg,
     1 hscale,hhh(2),adelh,linth,titlh
      character*56 title1,title2,title3
      character*20 titlx,titlz,titlg,titlh
      iplotr=1
      jplotr=iplotr
      xlimit=999.
      ylimit=999.
      do 1 i=1,2
      xxx(i)=0.0
      zzz(i)=0.0
      ggg(i)=0.0
1     hhh(i)=0.0
      sizea =0.08
      sizet =0.1
      sizel =0.15
      xscale=0.0
      zscale=0.0
      gscale=0.0
      hscale=0.0
      adelx =0.0
      adelz =0.0
      adelg =0.0
```

```
      adelh =0.0
      lintx=5
      lintz=5
      lintg=5
      linth=5
      titlx='kilometer'
      titlz='depth km'
      titlg='mGal'
      titlh='nTesla'
      return
      end
c*********************************************************************
      subroutine initm(iset)
c  Magnetic field parameters are stored: intensity, inclination,
c  declination where inc is deg positive down, and dec is deg positive
c  east of north.  Unit vector components are: x along profile, z
c  positive down, and y right cartesian.
      common /magxyz/ ef(3),ev(3),tdir(3),azmuth,icompn
      character ians*1,icom*1
      data d2r/1.745329e-2/
      ireset=iset
      if(ef(1).eq.0.0) go to 11
      if(ireset.eq.0) go to 100
c
5     type 10
10    format(' change earth s field vector ? '$)
      if(noyes(1).eq.0) go to 20
11    type *,' present earth s field vector '
      type *,'   intensity=',ef(1)
      type *,' inclination=',ef(2)
      type *,' declination=',ef(3)
      type *,' enter 3 new values '
      read(5,*,end=5,err=5) ef
c
20    type 25
25    format(' change the profile azimuth ? '$)
      if(noyes(1).eq.0) go to 30
      type *,' enter new azimuth'
      read(5,*,end=20,err=20) azmuth
c
30    icompn=0
      type  35
35    format(' do you want total field or a component :'$)
      read(5,37,end=999) ians
37    format(a16)
      call cvc(ians,2)
      if(ians(1:1).eq.'t') go to 100
      if(ians(1:1).ne.'c') go to 30
50    type  60
60    format(' x,y, or z component :'$)
      read(5,37,end=30) icom
      call cvc(icom,2)
      if(icom(1:1).eq.'x') icompn=1
      if(icom(1:1).eq.'y') icompn=2
```

```
      if(icom(1:1).eq.'z') icompn=3
      if(icompn.eq.0) go to 50
      type  70,icom(1:1)
70    format(' the calculated field values are now the ',a1,
     1       ' component',/,
     2 ' referenced to the profile direction as the x axis')
c
100   ei=ef(2)*d2r
      ed=(ef(3)-azmuth)*d2r
      eh=cos(ei)
      ev(1)=cos(ed)*eh
      ev(2)=sin(ed)*eh
      ev(3)=sin(ei)
      do 110 i=1,3
110   tdir(i)=ev(i)
      if(icompn.eq.0) return
      do 120 i=1,3
120   tdir(i)=0.
      if(icompn.eq.1) tdir(1)=1.
      if(icompn.eq.2) tdir(2)=1.
      if(icompn.eq.3) tdir(3)=1.
999   return
      end
c*********************************************************************
      subroutine pltdvr(ier)
c  plot driver for observed and calculated data
      common /field/  nf(10),f(2000)
      common /parm1/  nvert,nbody,nobs,mobs,grvswt,magswt,
     1          iave,jave,vex,istatn,iparsh,labelv,numbod
      common /plot1/  iplotr,sizea,sizet,sizel,
     1          title1,title2,title3,
     1          xscale,xxx(2),adelx1,lintx,itx(5),
     1          zscale,zzz(2),adelz1,lintz,itz(5),
     1          gscale,ggg(2),adelg1,lintg,itg(5),
     1          hscale,hhh(2),adelh1,linth,ith(5)
      common /window/ xwind(2,4),ywind(2,4)
      common /vuport/ xview(4,4),yview(4,4)
      common /scal1/  xp(4),yp(4),dxp(2),dyp(2)
      common /pltdev/ jplotr,xboard,yboard
c
      dimension nlab(3),it1(14)
      character title1*56,title2*56,title3*56,tit1*56
      logical grvswt,magswt
      equivalence (tit1,it1)
      data ix,iz,iobs,icalc,ioff/1,2,3,4,5/
c
c  initialize plot
      call clrscr(1)
      if(grvswt) mgswt=0
      if(magswt) mgswt=1
      if(grvswt .and. magswt) mgswt=-1
      startx=0.01
      starty=0.01
      nplot=2
```

```
      if(mgswt.eq.-1) nplot=3
      call setwin(mgswt,xxx,zzz,ggg,hhh)
      call openvu(0,ier)
      call setvu(startx,starty,nlab,nlabel,ier)
      if(ier.gt.0) then
        type *,'pltdvr:setvu error=',ier
        call waiter(2.0)
        go to 99
      endif
c
c  plot model
      call openvu(1,ier)
      if(ier.eq.0) call pltmod
c
c  gravity plot
      if(grvswt) then
      iview=2
      np=nobs
      call openvu(iview,ier)
      if(ier.eq.0) then
      ltyp=0
      call pltgrf(f(nf(ix)),f(nf(icalc)),np,ltyp,iview)
c     character 6 is a '+'
      ltyp=-6
      call pltgrf(f(nf(ix)),f(nf(iobs)),np,ltyp,iview)
      endif
      endif
c
c  magnetic plot
      if(magswt) then
      iview=3
      np=mobs
      call openvu(iview,ier)
      if(ier.eq.0) then
      ltyp=0
      call pltgrf(f(nf(ix+ioff)),f(nf(icalc+ioff)),np,ltyp,iview)
      ltyp=-11
c     character 11 is a triangle
      call pltgrf(f(nf(ix+ioff)),f(nf(iobs+ioff)),np,ltyp,iview)
      endif
      endif
c
c  main titles
      if(ier.eq.0) then
      xc=xp(3)+0.5*xp(1)
      yt=yp(3)+yp(1)+((nlabel-1)+0.5)*(1.5*sizel)
      size=sizel
      do i=1,3
      if(i.gt.1) size=.75*sizel
      if(nlab(i).gt.0) then
        if(i.eq.1) titl(1:56)=title1(1:56)
        if(i.eq.2) titl(1:56)=title2(1:56)
        if(i.eq.3) titl(1:56)=title3(1:56)
        xt=xc-0.5*float(nlab(i))*size
```

```
        call vchar(xt,yt,it1,nlab(i),3,size,0.0,0.0,0.0)
        yt=yt-1.5*sizel
      endif
      end do
      endif
c
c  close plot
99    call openvu(-1,ier)
      call clrscr(2)
      return
      end
c*********************************************************************
      subroutine isodvr
c  plot the calculated curves for isolated groups of bodies
      common /field/  nf(10),f(2000)
      common /swork/  a(16800)
      common /parm1/  nvert,nbody,nobs,mobs,grvswt,magswt,
     1               iave,jave,vex,istatn,iparsh,labelv,numbod
      common /window/ xwind(2,4),ywind(2,4)
      dimension includ(20)
      logical grvswt,magswt
      data itty/5/
c
      m=nobs+mobs
c  put curves in the work array
10    type *,' enter number of curves to be plotted (max=7)'
      read(itty,*,end=999,err=999) ncurv
      if(ncurv.lt.1 .or. ncurv.gt.7) go to 999
      do i=1,m*ncurv
        a(i)=0.0
      end do
      ig=1
      ih=nobs+1
      do icurv=1,ncurv
      il=icurv
      type *,' enter nbody and body indices for curve', icurv
      read(itty,*,end=10,err=10) n,(includ(i),i=1,n)
        do i=1,n
        if(grvswt) then
          call sumfwd(0,includ(i),nobs,a(ig))
          call ammi(nobs,a(ig),ywind(1,2),ywind(2,2),mn,mx,il)
        endif
        if(magswt) then
          call sumfwd(1,includ(i),mobs,a(ih))
          call ammi(mobs,a(ih),ywind(1,3),ywind(2,3),mn,mx,il)
        endif
        end do
      ig=ig+m
      ih=ih+m
      end do
c
      call plota(1,nobs,1,mobs,m,ncurv,a)
999   return
      end
```

```
c**********************************************************************
      subroutine sumfwd(mgswt,ibody,np,pf)
      common /field/  nf(10),f(2000)
      common /parm1/  nvert,nbody,nobs,mobs
      common /model/  xb(100),zb(100),rho(50),sus(50),pl(50,2),
     1               rem(50,3)
      common /vertx/  icrn(500),ncrn(150),loca(150),nbord,ibord(100)
      dimension xb1(100),zb1(100),pf(np)
      do i=1,ncrn(ibody)
        xb1(i)=xb(icrn(loca(ibody)+i))
        zb1(i)=zb(icrn(loca(ibody)+i))
      end do
      if(mgswt.eq.0) then
        rs=rho(ibody)
        ix=1
        iz=ix+nobs
        else
        rs=sus(ibody)
        ix=5*nobs+1
        iz=ix+mobs
      endif
      call pickmg(mgswt,ibody,rs,ncrn(ibody),xb1,zb1,
     1           np,f(ix),f(iz),pf)
      return
      end
c**********************************************************************
      subroutine advr(igs,ngrv,ihs,nmag,no,npart,a,e)
c  driver for the partial derivative and error vector plots
      common /field/  nf(10),f(2000)
      common /parm1/  nvert,nbody,nobs,mobs,grvswt,magswt,
     1               iave,jave,vex,idepth,iparsh,indiv,numbod
      common /freel/  ifill(4),nx,nz,nr,ns,ix(20),iz(20),ir(20),is(20)
      common /window/ xwind(2,4),ywind(2,4)
      dimension xx(2),zz(2),gg(2),hh(2)
      dimension a(no,npart),e(no)
      logical grvswt,magswt
c
      mgswt=0
      if(magswt) mgswt=1
      if(grvswt .and. magswt) mgswt=-1
c  set x window
      xx(1)= 1.0e5
      xx(2)=-1.0e5
      if(grvswt) then
        igx=nf(1)+(igs-1)
        xx(1)=f(igx)
        xx(2)=f(igx+ngrv-1)
      endif
      if(magswt) then
        ihx=nf(6)+(ihs-1)
        xx(1)=amin1 (f(ihx),xx(1))
        xx(2)=amax1 (f(ihx+nmag-1),xx(2))
      endif
c
```

```
c  x partials
      if(nx.gt.0) then
      do j=1,nx
      jl=j
      if(grvswt) call ammi(ngrv,a(1,j),gg(1),gg(2),
     1          mn,mx,jl)
      if(magswt) call ammi(nmag,a(ngrv+1,j),hh(1),hh(2),
     1          mn,mx,jl)
      enddo
      call setwin(mgswt,xx,zz,gg,hh)
      call plota(igs,ngrv,ihs,nmag,no,nx,a)
      endif
c
c  z partials
      if(nz.gt.0) then
      js=nx+1
      je=nx+nz
      do j=js,je
      jl=j
      if(grvswt) call ammi(ngrv,a(1,j),gg(1),gg(2),
     1          mn,mx,jl)
      if(magswt) call ammi(nmag,a(ngrv+1,j),hh(1),hh(2),
     1          mn,mx,jl)
      enddo
      call setwin(mgswt,xx,zz,gg,hh)
      call plota(igs,ngrv,ihs,nmag,no,nz,a(1,js))
      endif
c
c  density and susceptibility
      if(nr+ns.gt.0) then
      js=nx+nz+1
      do j=js,nx+nz+nr
      jl=j
      call ammi(ngrv,a(1,j),     gg(1),gg(2),mn,mx,jl)
      enddo
      do j=js+nr,npart
      call ammi(nmag,a(ngrv+1,j),hh(1),hh(2),mn,mx,jl)
      enddo
      call setwin(mgswt,xx,zz,gg,hh)
      nrs=nr+ns
      call plota(igs,ngrv,ihs,nmag,no,nrs,a(1,js))
      endif
c
c  error vector
      if(grvswt) call ammi(ngrv,e(1),     gg(1),gg(2),mn,mx,1)
      if(magswt) call ammi(nmag,e(ngrv+1),hh(1),hh(2),mn,mx,1)
      call setwin(mgswt,xx,zz,gg,hh)
      call plota(igs,ngrv,ihs,nmag,no,1,e)
      return
      end
c**********************************************************************
      subroutine plota(igs,ngrv,ihs,nmag,m,ncurv,a)
c  plot multiple curves located in 'a'.
c  the functional values in 'a' are column-wise arranged grv then mag
```

```
c  with location i=1 corresponding to igx or ihx.
c  igs,ihs refer to the starting locations in field common.
      common /field/  nf(10),f(2000)
      common /parm1/  nvert,nbody,nobs,mobs,grvswt,magswt,
     1               iave,jave,vex,idepth,iparsh,indiv,numbod
      common ifill(4),nx,nz,nr,ns,ix(20),iz(20),ir(20),is(20)
      dimension a(m,ncurv)
      logical grvswt,magswt
c
      if(ngrv+nmag.ne.m) return
      igx=nf(1)+(igs-1)
      ihx=nf(6)+(ihs-1)
      call clrscr(1)
      call openvu(0,ier)
      if(ier.eq.0) call setvu(startx,starty,nlab,nlabel,ier)
      if(ier.eq.0) call openvu(1,ier)
      if(ier.gt.1) go to 88
      call pltmod
      if(grvswt) then
        call openvu(2,ier)
        if(ier.gt.0) go to 88
        do j=1,ncurv
        ltyp=j-1
        call pltgrf(f(igx),a(1,j),ngrv,ltyp,2)
        enddo
      endif
      if(magswt) then
        call openvu(3,ier)
        if(ier.gt.0) go to 88
        ih=ngrv+1
        do j=1,ncurv
        ltyp=j-1
        call pltgrf(f(ihx),a(ih,j),nmag,ltyp,3)
        end do
      endif
88    call openvu(-1,ier)
      call clrscr(2)
99    return
      end
c**********************************************************************
      subroutine pltmod
      common /field/  nf(10),f(2000)
      common /model/  xb(100),zb(100),fill(701)
      common /vertx/  icrn(500),ncrn(150),loca(150),nbord,ibord(100)
      common /parm1/  nvert,nbody,nobs,mobs,grvswt,magswt,
     1          iave,jave,vex,istatn,iparsh,labelv,numbod
      common /plot1/  iplotr,sizea,sizet,sizel,
     1          it1(14),it2(14),it3(14),
     1          xscale,xxx(2),adelx,lintx,itx(5),
     1          zscale,zzz(2),adelz,lintz,itz(5),
     1          gscale,ggg(2),adelg,lintg,itg(5),
     1          hscale,hhh(2),adelh,linth,ith(5)
      common /scal1/  xp(4),yp(4),dxp(2),dyp(2)
c
```

```
      dimension zlmt(2),ify(5),xbl(100),zbl(100)
      logical grvswt,magswt
      character fmty*20,titlz*20,body*2
      equivalence (fmty,ify),(titlz,itz)
c
c  draw model cross-section
      xsca=xp(1)/(dxp(2)-dxp(1))
      ysca=yp(1)/(dyp(2)-dyp(1))
      szlabl=1.25*sizea
      sizekm=sizea/xsca
      do 30 j=1,nbody
      do 10  i=1,ncrn(j)
      i2=icrn(loca(j)+i)
      xbl(i)=xb(i2)
10    zbl(i)=zb(i2)
      il=ncrn(j)+1
      xbl(il)=xbl(1)
      zbl(il)=zbl(1)
      call line(xbl,zbl,il,0,0)
c
c  number bodies
      if(numbod.eq.1) then
        write(body,20) j
20      format(i2)
        zlmt(1)=dyp(2)
        zlmt(2)=dyp(1)
        nch=leftj(body)
        call inpoly(a,b,ncrn(j),xbl,zbl,dxp,zlmt,sizekm,nch,ier)
        if(ier.eq.0 .and. sizea.gt.0.0) then
c           convert location to plotter inches
          a=xp(3)+xsca*(a-dxp(1))
          b=yp(3)+ysca*(b-dyp(1))
          call text(a,b,body,szlabl,0.0)
          endif
      endif
30    continue
c
c  plot observation locations
      if(istatn.eq.1) then
        if(grvswt) then
        ich=6
        call symbol(f(nf(1)),f(nf(2)),nobs,ich,sizea,0.0)
        endif
        if(magswt) then
        ich=11
        call symbol(f(nf(6)),f(nf(7)),mobs,ich,sizea,0.0)
        endif
      endif
c
c  depth axis annotation
      adely=adelz
      call setax(dyp,adely,15,nchary,fmty)
      call yaxis(dyp,dxp,yp,adely,lintz,sizea,ify,nchary)
      if(iplotr.ne.1) then
```

```
        nch=leftj(titlz)
        yt=0.5*yp(1)+yp(3)-float(nch/2)*sizet
        xt=xp(3)-nchary*sizea-sizet
        call text(xt,yt,titlz,sizet,90.0)
      endif
c
c  label vertices with a diamond and number
      if(labelv.eq.1) then
      ich=2
      tqsz=0.75*sizea
      hsz =0.5 *sizea
        do iv=1,nvert
        call symbol(xb(iv),zb(iv),1,ich,hsz,0.0)
        a=xp(3)+xsca*(xb(iv)-dxp(1))
        b=yp(3)+ysca*(zb(iv)-dyp(1))
        write(body,20) iv
        b=b-tqsz
        call text(a,b,body,tqsz,0.0)
        end do
      endif
      return
      end
c**********************************************************************
      subroutine inpoly(a,b,n,x,y,xlmt,ylmt,size,nsym,ier)
c  place labeling box inside a polygon, clip at xlmt, ylmt
      dimension xlmt(2),ylmt(2),x(n),y(n)
      logical inside
      ier=1
      ntst=20
      hsize = size/2.0
      xsize = (nsym+1)*(5.0/7.0)*size
      call ammi(n,x,xmn,xmx,mn,mx,0)
      call ammi(n,y,ymn,ymx,mn,mx,0)
      if(xlmt(1).gt.xmn) xmn=xlmt(1)
      if(xlmt(2).lt.xmx) xmx=xlmt(2)
      if(ylmt(1).gt.ymn) ymn=ylmt(1)
      if(ylmt(2).lt.ymx) ymx=ylmt(2)
      if(xmx-xmn.le.0.0) return
      if(ymx-ymn.le.0.0) return
      as = (xmn+xmx)/2.0
      bs = (ymn+ymx)/2.0
      a=as
      b=bs
c
      dx=(xmx-xmn)/float(ntst-1)
      dist=0.0
      b1=bs-hsize
      b2=b1+size
      do itst=0,ntst
      dist=-sign(1.0,dist)*(float(itst)*dx)
      a=as+dist
      a1=a-hsize
      a2=a1+xsize
      if(inside(n,x,y,a1,b1) .and. inside(n,x,y,a2,b2)) go to 99
```

```
      end do
c
      dy=(ymx-ymn)/float(ntst-1)
      dist=0.0
      a1=as-hsize
      a2=a1+xsize
      do itst=0,ntst
      dist=-sign(1.0,dist)*(float(itst)*dy)
      b=bs+dist
      b1=b-hsize
      b2=b1+size
      if(inside(n,x,y,a1,b1) .and. inside(n,x,y,a2,b2)) go to 99
      end do
      return
99    ier=0
      return
      end
c**********************************************************************
      subroutine pltgrf(x,y,np,ltyp,iview)
c  line type = 0 annotates and draws the first curve,
c            > 0 draws curves with various dashing patterns
c            < 0 draws symbols
      common /parm1/  nvert,nbody,ifill(2),grvswt,magswt,
     1          iave,jave,vex,istatn,iparsh,labelv,numbod
      common /plot1/  iplotr,sizea,sizet,sizel,
     1          it1(14),it2(14),it3(14),
     1          xscale,xxx(2),adelx,lintx,titlx,
     1          zscale,zzz(2),adelz,lintz,titlz,
     1          gscale,ggg(2),adelg,lintg,titlg,
     1          hscale,hhh(2),adelh,linth,titlh
      common /scal1/  xp(4),yp(4),dxp(2),dyp(2)
      dimension ifx(5),ify(5)
      dimension x(np),y(np)
      character*20 fmtx,fmty,titlx,titlz,titlg,titlh,tx,ty
      logical grvswt,magswt
      equivalence (fmtx,ifx),(fmty,ify)
c
      if(ltyp) 300,100,200
100   adelx1=adelx
      adely=adelg
      linty=lintg
      tx(1:20)=titlx(1:20)
      if(iview.eq.3 .and. (grvswt.and.magswt)) tx=' '
      ty(1:20)=titlg(1:20)
      if(iview.eq.3) then
      adely=adelh
      linty=linth
      ty(1:20)=titlh(1:20)
      endif
c
      call setax(dyp,adely, 30,nchary,fmty)
      call yaxis(dyp,dxp,yp,adely, linty,sizea, ify,nchary)
      nchy=leftj(ty)
      if(nchy.gt.0) then
```

```
        xt=xp(3)-nchary*sizea-sizet
        yt=yp(3)+0.5*yp(1)-(nchy/2)*sizet
        call text(xt,yt,ty,sizet,90.0)
      endif
c
      call setax(dxp,adelx1,50,ncharx,fmtx)
      if(iview.eq.3 .and. grvswt) ncharx=0
      call xaxis(dxp,dyp,xp,adelx1,lintx,sizea,ifx,ncharx)
      nchx=leftj(tx)
      if(nchx.gt.0) then
        xt2=xp(3) + 0.5*xp(1) - (nchx/2)*sizet
        yt2=yp(3)-ncharx*sizea-sizet
        call text(xt2,yt2,tx,sizet,0.0)
      endif
c
c  plot calculated function
200   ltyp1=mod(ltyp,8)
      call line(x,y,np,0,ltyp1)
      return
c
c  plot observed data
300   ich=iabs(ltyp)
      call symbol(x,y,np,ich,sizea,0.0)
      return
      end
c**********************************************************************
      subroutine setwin(mgswt,xxx,zzz,ggg,hhh)
c  assign window limits either from data or variables xxx,zzz,ggg,hhh.
c  the triple letter input variables have priority.
c  mgswt=-1 both fields, =0 gravity only, =1 magnetics only.
      common /field/  nf(10),f(2000)
      common /model/  xbody(100),ybody(100),fill(500)
      common /vertx/  fill2(800),nbord,ibord(100)
      common /parm1/  ifill1(2),nobs,mobs,ifill2(5),istatn,ifill3(3)
      common /window/ xwind(2,4),ywind(2,4)
      dimension xxx(2),zzz(2),ggg(2),hhh(2)
      data ix,iz,iobs,icalc,ioff /1,2,3,4,5/
c
      do j=1,3
        xwind(1,j)= 1.e10
        xwind(2,j)=-1.e10
        ywind(1,j)= 1.e10
        ywind(2,j)=-1.e10
      end do
c
c  x windows
      if(mgswt.lt.1 .and. nobs.gt.0)
     1  call ammi(nobs,f(nf(ix)),     xwind(1,2),xwind(2,2),mn,mx,1)
      if(mgswt.ne.0 .and. mobs.gt.0)
     1  call ammi(mobs,f(nf(ix+ioff)),xwind(1,3),xwind(2,3),mn,mx,1)
c
      if((xxx(1).ne.0.0 .or. xxx(2).ne.0.0) .and.
     1  xxx(1).lt.xxx(2)) then
        xwind(1,1)=xxx(1)
```

```
        xwind(2,1)=xxx(2)
      else
        xwind(1,1)=amin1(xwind(1,2), xwind(1,3))
        xwind(2,1)=amax1(xwind(2,2), xwind(2,3))
      endif
c
      do j=2,3
        xwind(1,j)=xwind(1,1)
        xwind(2,j)=xwind(2,1)
      end do
c
c  y windows
c     gravity
      if(mgswt.lt.1 .and. nobs.gt.0) then
        if(istatn.eq.1)  call ammi(nobs,f(nf(iz)),
     1                 ywind(1,1),ywind(2,1),mn,mx,2)
        call ammi(nobs,f(nf(iobs)),  ywind(1,2),ywind(2,2),mn,mx,2)
        call ammi(nobs,f(nf(icalc)), ywind(1,2),ywind(2,2),mn,mx,2)
      endif
c     magnetics
      if(mgswt.ne.0 .and. mobs.gt.0) then
        if(istatn.eq.1) call ammi(mobs,f(nf(iz+ioff)),
     1                 ywind(1,1),ywind(2,1),mn,mx,2)
        call ammi(mobs,f(nf(iobs+ioff)),  ywind(1,3),ywind(2,3),mn,mx,2)
        call ammi(mobs,f(nf(icalc+ioff)), ywind(1,3),ywind(2,3),mn,mx,2)
      endif
c    model
      do k=1,nbord
        yb=ybody(ibord(k))
        if(yb.lt.ywind(1,1)) ywind(1,1)=yb
        if(yb.gt.ywind(2,1)) ywind(2,1)=yb
      end do
c    minimum range
      do j=2,3
        yrange=abs(ywind(2,j)-ywind(1,j))
        if(yrange.eq.0.0) yrange=1.0
        ywind(2,j)=ywind(2,j)+.05*yrange
        ywind(1,j)=ywind(1,j)-.05*yrange
      end do
c
c  specified limits
      if((zzz(1).ne.0.0 .or. zzz(2).ne.0.0)) then
        ywind(1,1)=zzz(1)
        ywind(2,1)=zzz(2)
      endif
      if((ggg(1).ne.0.0 .or. ggg(2).ne.0.0) .and.
     1  ggg(1).lt.ggg(2)) then
        ywind(1,2)=ggg(1)
        ywind(2,2)=ggg(2)
      endif
      if((hhh(1).ne.0.0 .or. hhh(2).ne.0.0) .and.
     1  hhh(1).lt.hhh(2)) then
        ywind(1,3)=hhh(1)
        ywind(2,3)=hhh(2)
```

```
      endif
c
c  ad hoc... z positive down in plot area
      if(ywind(1,1).lt.ywind(2,1)) then
        tmp=ywind(1,1)
        ywind(1,1)=ywind(2,1)
        ywind(2,1)=tmp
      endif
      return
      end
c**********************************************************************
      subroutine setvu(startx,starty,nlab,nlabel,ier)
c  Set dimensions for the cross-section, gravity, and magnetic
c  viewports.  When the scaling defaults, the associated viewport is
c  a set fraction of the total device width.  Viewport parameters in
c  plotter units: 1,2 are minimum,maximum coordinates
c  3,4 are left,right interior margin widths for a scaled data area.
      common /parm1/  nvert,nbody,nobs,mobs,grvswt,magswt,
     1          iave,jave,vex,istatn,iparsh,labelv,numbod
      common /plot1/  iplotr,sizea,sizet,sizel,
     1          title1,title2,title3,
     1          xscale,xxx(2),adelx,lintx,itx(5),
     1          zscale,zzz(2),adelz,lintz,itz(5),
     1          gscale,ggg(2),adelg,lintg,itg(5),
     1          hscale,hhh(2),adelh,linth,ith(5)
      common /pltdev/ jplotr,xboard,yboard
      common /window/ xwind(2,4),ywind(2,4)
      common /vuport/ xview(4,4),yview(4,4)
      character title1*56,title2*56,title3*56
      logical grvswt,magswt
      dimension scale(3),nlab(3)
c
      nport=3
      nplot=2
      if(grvswt .and. magswt) nplot=3
      ier=1
      if(xboard.le.0.0 .or. yboard.le.0.0) return
      do i=1,nport
        scale(i)=0.0
      end do
      xsca=0.0
      if(xscale.gt.0.0) xsca=    xscale
      if(zscale.gt.0.0) scale(1)=zscale
      if(gscale.gt.0.0) scale(2)=gscale
      if(hscale.gt.0.0) scale(3)=hscale
c
c  labelling space
      sizet1=sizet
      sizel1=sizel
      nchar=13
      nlabel=0
      if(iplotr.eq.1) then
c   conserve space on video terminals
        sizet1=0.0
```

```
        sizel1=0.0
        nchar=7
      else
        nlab(1)=leftj(title1)
        nlab(2)=leftj(title2)
        nlab(3)=leftj(title3)
        do i=1,3
        if(nlab(i).gt.0) nlabel=nlabel+1
        end do
      endif
      xymarg=3.0*sizetl+nchar*sizea
      if(xymarg.lt.0.0) xymarg=0.0
      startx=0.02*xboard
      starty=0.02*yboard
c
c  x dimensions
      if(2.0*startx.gt.xboard) startx=0.0
      do 10 j=1,3
      xview(1,j)=startx
      xview(2,j)=xboard-startx
      xview(3,j)=xymarg
      xview(4,j)=sizea
      do 10 i=1,4
10    yview(i,j)=0.0
c
c  rightside margins with scaling included
      wdata=abs(xwind(2,1)-xwind(1,1))
      if(xsca.gt.0.0) then
        xpl = wdata/xsca
        xview(4,1) = xview(2,1)-(xview(1,1)+xview(3,1)+xpl)
        if(xview(4,1).lt.0.0) then
          xview(4,1)=sizea
          xwind(2,1) = xsca*pl(xview(1,1)) + xwind(1,1)
        endif
      else
        ier=2
        if(pl(xview(1,1)).le.0.0) return
        if(xsca.eq.0.0) xsca=wdata/pl(xview(1,1))
        ier=3
        if(xsca.le.0.0) return
      endif
c
      do 20 j=2,3
      xwind(2,j)=xwind(2,1)
      do 20 i=1,4
20    xview(i,j)=xview(i,1)
      if(scale(1).le.0.0) scale(1)=xsca
      if(vex.le.0.0) vex=1.0
      scale(1)=scale(1)/vex
c
c  model cross-section dimensions
c
      ydef=(yboard-2.*starty)/float(nport)
      ysep=0.1*ydef
```

```
      wdata=abs(ywind(2,1)-ywind(1,1))
c
c  viewport size
      yview(1,1)=starty
      yview(2,1)=ydef
      yview(3,1)=0.0
      yview(4,1)=ysep
      if(xscale.gt.0.0 .or. zscale.gt.0.0) then
        yview(2,1) = yview(1,1)+ysep+wdata/scale(1)
        ylimit = 0.7*(yboard-2.0*starty)
        if(yview(2,1)-yview(1,1).gt.ylimit) yview(2,1)=ylimit
      endif
c
c  cutoff bottom of the window to maintain scaling
      yview(3,1)=yview(2,1)-yview(1,1)-yview(4,1)-wdata/scale(1)
      if(yview(3,1).lt.0.0) then
        yview(3,1)=0.0
        ywind(1,1)=ywind(2,1)+scale(1)*pl(yview(1,1))
      endif
c
c  graph dimensions, port 2 is gravity, port 3 is magnetics
c
      ydef=((yboard-starty)-yview(2,1))/(nplot-1)
      do 50 j=2,nport
      yview(1,j)=yview(2,j-1)
      yview(2,j)=yview(1,j)
      if(j.eq.2 .and. .not.grvswt) go to 50
      if(j.eq.3 .and. .not.magswt) go to 50
c
      if(j.eq.3 .and. grvswt) xymarg=0.0
      if(j.eq.3 .or. (j.eq.2 .and. .not.magswt))
     1  ysep=nlabel*(1.5*sizel)
      yview(3,j)=xymarg
      yview(4,j)=ysep
c
c  viewport size
      wdata=abs(ywind(2,j)-ywind(1,j))
      if(scale(j).gt.0.0) then
        yview(2,j)=yview(1,j)+yview(3,j)+yview(4,j)+wdata/scale(j)
        if(yview(2,j).gt.yboard-starty) yview(2,j)=yboard-starty
        else
        yview(2,j)=yview(1,j)+ydef
      endif
c
      ier=4
      ypl=pl(yview(1,j))
      if(ypl.le.0.0) then
        yview(3,j)=0.0
        yview(4,j)=0.0
        ypl=pl(yview(1,j))
      endif
      if(ypl.lt.0.0) return
c  adjust window if necessary to maintain specified scaling
      if(scale(j).gt.0.0 .and. ypl.ne.wdata/scale(j))
```

```
     1  ywind(1,j)=ywind(2,j)-scale(j)*yp1
c
50     continue
       ier=0
       return
       end
c**********************************************************************
       function p1(a)
       dimension a(4)
       p1=a(2)-a(1)-a(3)-a(4)
       return
       end
c**********************************************************************
       subroutine openvu(iview,ier)
c  translate window and viewport info into the plot system form
c  and perform a scaling call to activate the viewport.
c
c  short summary of the plot system by G.I. Evenden and R.R. Wahl USGS.
c
c    >>pltset(jplotr,xboard,yboard,1) initializes device in inches.
c  x & y board are returned plotter dimensions.
c
c    >>scale(dxp,dyp,xp,yp,nopts,ierror)
c  Definition of plot area: xp(1:4),yp(1:4) units are inches:
c  xp(1)= data area width, 2=0 is linear scale, 3=left margin,
c  4=plotter width.
c  Definition of data area: dxp(1:2),dyp(1:2) in data units:
c  dxp(1)=minimum x coordinate, dxp(2)=maximum x coordinate.
c  Vectors are clipped at the data area boundary.
c
c    >>line(x,y,n,icon,ipen) where x,y are data unit arrays n long,
c  icon.ne.0 starts new line, and ipen selects dashing pattern.
c    >>xaxis,yaxis axis tic routines
c    >>endpt closes out the system
c
c  viewport coordinates in inches
c  window coordinates in data units (kilometers, mgal, etc)
c
       common /pltdev/ jplotr,xboard,yboard,xlimit,ylimit
       common /window/ xwind(2,4),ywind(2,4)
       common /vuport/ xview(4,4),yview(4,4)
       common /scall/  xvu(4),yvu(4),xdata(2),ydata(2)
       dimension xp(4),yp(4)
c
       if(iview.lt.0) then
         call endpt(0)
         xboard=0.0
         yboard=0.0
         return
       endif
c
       if(iview.eq.0) then
c  initialize plot system, return device dimensions
         call pltset(jplotr,xboard,yboard,1)
```

```
c  normally limit electrostatic plotter size to one paper width.
        if(jplotr.eq.6 .and.
     1       (xlimit.gt.xboard .or. ylimit.gt.yboard)) then
          xlimit=72.
          ylimit=20.
        endif
        if(xlimit.lt.xboard) xboard=xlimit
        if(ylimit.lt.yboard) yboard=ylimit
        return
      endif
c
c  open viewport
      call getvu(xview(1,iview),yview(1,iview),xp,yp,ier)
      if(ier.ne.0) type *,' openvu:getvu error number', ier
      if(ier.gt.0) return
      ier=0
      call scale(xwind(1,iview),ywind(1,iview),xp,yp,4,ier)
      if(ier.ne.0) then
        type *,'openvu: scaling error'
        return
      endif
      do 5 i=1,4
      xvu(i)=xp(i)
5     yvu(i)=yp(i)
      do 10 i=1,2
      xdata(i)=xwind(i,iview)
10    ydata(i)=ywind(i,iview)
      return
      end
c**********************************************************************
      subroutine getvu(xview,yview,xp,yp,ier)
c  input viewport data, returns 4 parameter data area specification
c  all values are positive and in plot units
      common /pltdev/ jplotr,xboard,yboard
      dimension xview(4),yview(4),xp(4),yp(4)
      tol=1.e-4
      xp(2)=0.0
      yp(2)=0.0
      ier=10000
      xp(4)=xboard
      yp(4)=yboard
      if(xp(4).le.0.0 .or. yp(4).le.0.0) return
      ier=1
      xwidth = xview(2)-xview(1)
      if(xwidth.lt.0.0) go to 50
      ier=2
      xp(3) = xview(1) + xview(3)
      if(xp(3).lt.0.0) return
      ier=3
      xp(1) = xwidth - (xview(3)+xview(4))
      if(xp(1).lt.0.0) go to 50
      ier=0
c
50    ier=ier+10
```

```
      ywidth = yview(2)-yview(1)
      if(ywidth.lt.0.0) go to 99
      ier=ier+20
      yp(3) = yview(1) + yview(3)
      if(yp(3).lt.0.0) go to 99
      yp(1) = ywidth - (yview(3)+yview(4))
      ier=ier+30
      if(yp(1).lt.0.0) go to 99
      ier=0
c
99    if(xp(3).gt.tol) xp(3)=xp(3)-tol
      if(yp(3).gt.tol) yp(3)=yp(3)-tol
      if(xp(1)+xp(3).gt.xp(4) .and. xp(1).gt.tol) xp(1)=xp(1)-tol
      if(yp(1)+yp(3).gt.yp(4) .and. yp(1).gt.tol) yp(1)=yp(1)-tol
      xptest=xp(4)-xp(1)-xp(3)
      if(xptest.lt.0.0) then
        xp(1)=xp(4)
        xp(3)=0.0
        ier=ier+100
      endif
      yptest=yp(4)-yp(1)-yp(3)
      if(yptest.lt.0.0) then
        yp(1)=yp(4)
        yp(3)=0.0
        ier=ier+1000
      endif
      return
      end
c**********************************************************************
      subroutine setax(x,dx,maxint,nch,fmt)
c  adjust interval and labeling format
c  input x-min max range, optional dx-spacing and maxint-intervals
c  returns dx, nch-number of significant figures, fmt-labeling format
      dimension x(2)
      character fmt*(*)
      ixpn(r)=int(alog10(abs(r))+100.)-100
      fmt='(1pe13.6)'
      nch=13
      if(maxint.le.0) maxint=20
      dt=abs(x(2)-x(1))/float(maxint)
      if(dt.lt.1.e-20 .and. dt.gt.-1.e-20) go to 20
      if(dx.ne.0.) go to 5
      p10=sign(1.0,dt)*10.**ixpn(dt)
      t1=dt/p10
      if(t1.le.1.0) dx=p10
      if(t1.gt.1.0) dx=2.*p10
      if(t1.gt.2.0) dx=5.*p10
      if(t1.gt.5.0) dx=10.*p10
5     idecm=0
      m10=ixpn(dx)
      if(m10.lt.0) idecm=iabs(m10)
      iw=4+idecm
      m20=ixpn( amax1(abs(x(1)),abs(x(2))) )
      if(m20.ge.1) iw=iw+m20
```

```
      if(iw.gt.9 .or. idecm.gt.9) go to 11
      write(fmt,10) iw,idecm
10    format('(f',i1,'.',i1,')' )
      nch=iw
      if(idecm.eq.0) nch=nch-1
11    return
20    if(dx.eq.0.) dx=1.e20
      return
      end
c**********************************************************************
      subroutine adnois(nf,f,a,grvswt,magswt)
      dimension nf(10),a(400),f(1)
      character ifun*20
      logical grvswt,magswt
      nobs=nf(2)-nf(1)
      mobs=nf(7)-nf(6)
c
340   type  341
341   format(' add gaussian noise to calculated or observed :'$)
      read(5,37,end=999) ifun
37    format(a20)
      call cvc(ifun,2)
      iswt=-1
      if(ifun(1:2).eq.'ob') iswt=0
      if(ifun(1:2).eq.'ca') iswt=1
      if(iswt.eq.-1) go to 349
3400  type  342
342   format(' enter percent error :'$)
      read(5,*,end=3400,err=3400) perc
      type 343
343   format(' correlate the noise ?'$)
      icorr=noyes(ldum)
      if(icorr.eq.1) then
345   type 346
346   format(' enter correlation width (samples) :'$)
      read(5,*,end=345,err=345) iw
      if(iw.lt.2) iw=2
      endif
      noff=0
      n=nobs
      if(grvswt) go to 351
350   noff=5
      n=mobs
351   iadr=nf(3+iswt+noff)
      call ammi(n,f(iadr),fmn,fmx,mn,mx,1)
      sd=0.5*(perc/100.)*(fmx-fmn)
      if(sd.lt.1.e-20) then
      type *,' zero range function'
      go to 349
      endif
      do 354 i=1,n+iw
      call gaussd(seed,sd,0.,a(i))
354   continue
      if(icorr.eq.1) then
```

```
      t=0.0
      do 360 i=1,iw
360   t=t+a(i)
      do 370 i=1,n
      ave=t/float(iw)
      t=t-a(i)+a(i+iw)
370   a(i)=ave
      endif
      iadr1=iadr-1
      do 380 i=1,n
380   f(iadr1+i)=f(iadr1+i)+a(i)
      if(magswt .and. noff.eq.0) go to 350
      return
349   type  348
348   format(' no change')
999   return
      end
c*********************************************************************
      subroutine gaussd(seed,sd,xmean,v)
      save iseed
      data is/1111111111/
      if(iseed.eq.0) iseed=is
      a=0.
      do 1 i=1,12
1     a=a+ran(iseed)
      v=(a-6.0)*sd+xmean
      return
      end
c*********************************************************************
      subroutine trend(x,y,n,dc,dy)
      dimension x(n),y(n)
      dy1=(y(n)-y(1))/(x(n)-x(1))
      dc1=y(1)
      type  10,dc1,dy1
10    format(' dc and slope to zero ends:',2f10.4)
      call fit1(x,y,n,0.,dy2,dc2)
      type  11,dc2,dy2
11    format(' dc and slope to fit least square :',2f10.4)
      type  12
12    format(' enter desired dc and slope for removal ')
      read(5,*,end=999,err=999) dc,dy
      do 20 i=1,n
20    y(i)=y(i)-dc-(x(i)-x(1))*dy
      return
999   type  98
98    format(' no change')
      return
      end
c*********************************************************************
      subroutine drpsur(xo,dx,nobs,z1, xf,zf)
      common /vertx/ ifill(800), nbord, ibord(100)
      common /model/ xb(100),zb(100)
      dimension xf(nobs),zf(nobs)
      if(nbord.le.0) go to 999
```

```
100   type *,' enter  start, ending vertices '
      type 1
1     format(' defining the top surface: '$)
      read(5,*,end=100,err=100) istart,iend
      zl = abs(zl)
c
c  js,je are ibord indices corresponding to vertices istart,iend.
      js = 0
      je = 0
      do i = 1, nbord
        if (ibord(i).eq.istart) js=i
        if (ibord(i).eq.iend)   je=i
      enddo
c
      do i = js, je-1
      if(xb(ibord(i+1)).lt.xb(ibord(i))) then
        type *,' top surface needs to be increasing in x'
        return
      endif
      enddo
c
      do 40 i = 1, nobs
      xf(i) = xo + dx*float(i-1)
      zf(i) = 0.0
      if       (xf(i).lt.xb(ibord(js))) then
        zf(i) = zb(ibord(js)) - zl
        elseif (xf(i).gt.xb(ibord(je))) then
        zf(i) = zb(ibord(je)) - zl
      endif
      if (xf(i).ge.xb(ibord(js)) .and. xf(i).le.xb(ibord(je))) then
      j = js
30    if (j.ge.je) go to 40
      k  = ibord(j)
      k1 = ibord(j+1)
      if (xf(i).ge.xb(k) .and. xf(i).lt.xb(k1)) then
        xd = xb(k1) - xb(k)
        if (xd.le.1.e-20) go to 40
        zd = zb(k1) - zb(k)
        x1 =  xf(i) - xb(k)
        zf(i) = (x1*zd)/xd + (zb(k)-zl)
      else
        j=j+1
        go to 30
      endif
      endif
40    continue
c
      return
999   type  998
998   format(' no boundary array available, cannot drape')
      return
      end
c*********************************************************************
      subroutine svd(a,m,n,s,u,v,wrk)
```

```
      dimension a(m,n),s(n),u(m,n),v(n,n),wrk(1)
      n2=n+1
      n3=n+n2
      call svd1(a,m,n,m,n,0,n,n,s,u,v,wrk,wrk(n2),wrk(n3))
      return
      end
      subroutine svd1(a,mmax,nmax,m,n,p,nu,nv,s,u,v,b,c,t)
c  singular value decomposition of a matrix
c  Peter A. Businger, Bell Telephone Laboratories
c  Gene H. Golub, Stanford University
c  Algorithm 358, Collected algorithms from ACM
c  modified for real matrix by Mike Webring, USGS
      dimension a(mmax,1), u(mmax,1), v(nmax,1)
      dimension s(n), b(n), c(n), t(n)
      integer p
      data eta,tol/1.5e-8, 1.e-31/
      np=n+p
      n1=n+1
c
c  householder reduction
      c(1)=0.0
      k=1
10    k1=k+1
c
c  elimination of a(i,k), i=k+1...m
      z=0.0
      do 20 i=k,m
20    z = z + a(i,k)**2
      b(k)=0.0
      if(z.le.tol) go to 70
      z=sqrt(z)
      b(k)=z
      w=abs(a(k,k))
      q=1.0
      if(w.ne.0.0) q=a(k,k)/w
      a(k,k)=q*(z+w)
      if(k.eq.np) go to 70
      do 50 j=k1,np
      q=0.0
      do 30 i=k,m
30    q=q+a(i,k)*a(i,j)
      q=q/(z*(z+w))
      do 40 i=k,m
40    a(i,j)=a(i,j)-q*a(i,k)
50    continue
c
c  phase transformation
      q=-sign(1.0,a(k,k))
      do 60 j=k1,np
60    a(k,j)=q*a(k,j)
c
c  elimination of a(k,j), j=k+2...n
70    if(k.eq.n) go to 140
      z=0.0
```

```
      do 80 j=k1,n
80    z = z + a(k,j)**2
      c(k1)=0.0
      if(z.le.tol) go to 130
      z=sqrt(z)
      c(k1)=z
      w=abs(a(k,k1))
      q=1.0
      if(w.ne.0.0) q=a(k,k1)/w
      a(k,k1)=q*(z+w)
      do 110 i=k1,m
      q=0.0
      do 90 j=k1,n
90    q = q + a(k,j)*a(i,j)
      q=q/(z*(z+w))
      do 100 j=k1,n
100   a(i,j)=a(i,j)-q*a(k,j)
110   continue
c
c  phase transformation
      q=-sign(1.0,a(k,k1))
      do 120 i=k1,m
120   a(i,k1)=a(i,k1)*q
130   k=k1
      go to 10
c
c  tolerance for negligible elements
140   eps=0.0
      do 150 k=1,n
      s(k)=b(k)
      t(k)=c(k)
150   eps=amax1(eps,s(k)+t(k))
      eps=eps*eta
c
c  initialization of u and v
      if(nu.eq.0) go to 180
      do 170 j=1,nu
      do 160 i=1,m
160   u(i,j)=0.0
170   u(j,j)=1.0
180   if(nv.eq.0) go to 210
      do 200 j=1,nv
      do 190 i=1,n
190   v(i,j)=0.0
200   v(j,j)=1.0
c
c  qr diagonalization
210   do 380 kk=1,n
      k=n1-kk
c
c  test for split
220   do 230 mm=1,k
      m2=k+1-mm
      if(abs(t(m2)).le.eps) go to 290
```

```
      if(abs(s(m2-1)).le.eps) go to 240
230   continue
c
c  cancellation of e(m2)
240   cs=0.0
      sn=1.0
      m1=m2-1
      do 280 i=m2,k
      f=sn*t(i)
      t(i)=cs*t(i)
      if(abs(f).le.eps) go to 290
      h=s(i)
      w=sqrt(f*f+h*h)
      s(i)=w
      cs=h/w
      sn=-f/w
      if(nu.eq.0) go to 260
      do 250 j=1,n
      x=u(j,m1)
      y=u(j,i)
      u(j,m1) = x*cs + y*sn
250   u(j,i)  = y*cs - x*sn
260   if(np.eq.n) go to 280
      do 270 j=n1,np
      q=a(m1,j)
      r=a(i,j)
      a(m1,j) = q*cs + r*sn
270   a(i,j)  = r*cs - q*sn
280   continue
c
c  test for convergence
290   w=s(k)
      if(m2.eq.k) go to 360
c
c  origin shift
      x=s(m2)
      y=s(k-1)
      g=t(k-1)
      h=t(k)
      f=((y-w)*(y+w)+(g-h)*(g+h))/(2.0*h*y)
      g=sqrt(f*f+1.0)
      if(f.lt.0.0) g=-g
      f=((x-w)*(x+w)+(y/(f+g)-h)*h)/x
c
c  qr step
      cs=1.0
      sn=1.0
      m1=m2+1
      do 350 i=m1,k
      g=t(i)
      y=s(i)
      h=sn*g
      g=cs*g
      w=sqrt(h*h+f*f)
```

```
      t(i-1)=w
      cs=f/w
      sn=h/w
      f=x*cs+g*sn
      g=g*cs-x*sn
      h=y*sn
      y=y*cs
      if(nv.eq.0) go to 310
      do 300 j=1,n
      x=v(j,i-1)
      w=v(j,i)
      v(j,i-1)= x*cs + w*sn
300   v(j,i)  = w*cs - x*sn
310   w=sqrt(h*h+f*f)
      s(i-1)=w
      cs=f/w
      sn=h/w
      f=cs*g+sn*y
      x=cs*y-sn*g
      if(nu.eq.0) go to 330
      do 320 j=1,n
      y=u(j,i-1)
      w=u(j,i)
      u(j,i-1) = y*cs + w*sn
320   u(j,i)   = w*cs - y*sn
330   if(n.eq.np) go to 350
      do 340 j=n1,np
      q=a(i-1,j)
      r=a(i,j)
      a(i-1,j)=q*cs+r*sn
340   a(i,j) = r*cs-q*sn
350   continue
      t(m2)=0.0
      t(k)=f
      s(k)=x
      go to 220
c
c  convergence
360   if(w.ge.0.0) go to 380
      s(k)=-w
      if(nv.eq.0) go to 380
      do 370 j=1,n
370   v(j,k)=-v(j,k)
380   continue
c
c  sort singular values
      do 450 k=1,n
      g=-1.0
      j=k
      do 390 i=k,n
      if(s(i).le.g) go to 390
      g=s(i)
      j=i
390   continue
```

```
      if(j.eq.k) go to 450
      s(j)=s(k)
      s(k)=g
      if(nv.eq.0) go to 410
      do 400 i=1,n
      q=v(i,j)
      v(i,j)=v(i,k)
400   v(i,k)=q
410   if(nu.eq.0) go to 430
      do 420 i=1,n
      q=u(i,j)
      u(i,j)=u(i,k)
420   u(i,k)=q
430   if(n.eq.np) go to 450
      do 440 i=n1,np
      q=a(j,i)
      a(j,i)=a(k,i)
440   a(k,i)=q
450   continue
c
c  back transformation
      if(nu.eq.0) go to 510
      do 500 kk=1,n
      k=n1-kk
      if(b(k).eq.0.0) go to 500
      q=-sign(1.0,a(k,k))
      do 460 j=1,nu
460   u(k,j)=q*u(k,j)
      do 490 j=1,nu
      q=0.0
      do 470 i=k,m
470   q = q + a(i,k)*u(i,j)
      q = q/(abs(a(k,k))*b(k))
      do 480 i=k,m
480   u(i,j)=u(i,j)-q*a(i,k)
490   continue
500   continue
510   if(nv.eq.0) go to 570
      if(n.lt.2) go to 570
      do 560 kk=2,n
      k=n1-kk
      k1=k+1
      if(c(k1).eq.0.0) go to 560
      q=-sign(1.0,a(k,k1))
      do 520 j=1,nv
520   v(k1,j)=q*v(k1,j)
      do 550 j=1,nv
      q=0.0
      do 530 i=k1,n
530   q=q+a(k,i)*v(i,j)
      q=q/(abs(a(k,k1))*c(k1))
      do 540 i=k1,n
540   v(i,j)=v(i,j)-q*a(k,i)
550   continue
```

```
560   continue
570   return
      end
c*************************************************************************
      subroutine ata(a,m,n,b)
      dimension a(m,n),b(n,n)
      do 10 j=1,n
      do 10 i=1,n
10    b(i,j)=xty(a(1,i),1,a(1,j),1,m)
      return
      end
c*************************************************************************
      subroutine axb(a,ma,na,b,mb,nb,c)
      dimension a(ma,na),b(mb,nb),c(ma,nb)
      if(na.ne.mb) go to 20
      k=na
      do 10 j=1,nb
      do 10 i=1,ma
10    c(i,j)=xty(a(i,1),ma,b(1,j),1,k)
      return
20    type 21
21    format(' axb: col a .ne. row b')
      return
      end
c*************************************************************************
      subroutine atb(a,ma,na,b,mb,nb,c)
      dimension a(ma,na),b(mb,nb),c(na,nb)
      if(ma.ne.mb) go to 20
      k=ma
      do 10 j=1,nb
      do 10 i=1,na
10    c(i,j)=xty(a(1,i),1,b(1,j),1,k)
      return
20    type 21
21    format(' atb: col.ne.row')
      return
      end
c*************************************************************************
      function xty(x,ix,y,iy,n)
      dimension x(n),y(n)
      double precision tmp,tp,tn
      tp=0.0d0
      tn=0.0d0
      jx=1
      jy=1
      do 10 i=1,n
      tmp=dble(x(jx))*dble(y(jy))
      if(tmp.lt.0.0d0) go to 5
      tp=tp+tmp
      go to 6
5     tn=tn+tmp
6     jx=jx+ix
10    jy=jy+iy
      xty=sngl(tp+tn)
```

```
      return
      end
c*********************************************************************
      subroutine chki(a,n,iexp)
c  check the identity matrix
      dimension a(n,n)
      iexp=15
      tmx=0.0
      do 10 j=1,n
      do 10 i=1,n
      t=abs(a(i,j))
      if(i.eq.j) t=abs(1.0-a(i,i))
      if(t.gt.tmx) tmx=t
10    continue
      if(tmx.le.1.e-15) return
      iexp=int(0.5-alog10(tmx))
      return
      end
c*********************************************************************
      subroutine fit1(xa,y,n,dx,a,y0)
c  least square fit to  y=ax+y0
      dimension z(5),y(n),xa(1)
      equivalence (z(1),sx),(z(2),sy),(z(3),sxy),
     1 (z(4),sx2),(z(5),x)
      do 1 i=1,5
1     z(i)=0.
      ix=0
      if(dx.eq.0.) ix=1
      a=0.
      y0=y(1)
      if(n.le.1) return
      do 2 i=1,n
      if(ix.gt.0) x=xa(i)-xa(1)
      sx=sx+x
      sy=sy+y(i)
      sxy=sxy+x*y(i)
      sx2=sx2+x*x
2     x=x+dx
      xn=float(n)
      d=xn*sx2-sx*sx
      a=1.e10
      if(abs(d) .lt. 1.e-10) return
      a=(xn*sxy-sx*sy)/d
      y0=(sy-a*sx)/xn
      return
      end
c*********************************************************************
      function correl(n,a,b)
      dimension a(n),b(n)
      correl=0.0
      ab=0.0
      a2=0.0
      b2=0.0
      do 1 i=1,n
```

```
      ab=ab+a(i)*b(i)
      a2=a2+a(i)*a(i)
1     b2=b2+b(i)*b(i)
      a2b2=a2*b2
      if(a2b2.lt.1.e-30) return
      correl=abs(ab)/sqrt(a2b2)
      return
      end
c*****************************************************************************
      function acosd2(dx,dh)
      acosd2=1.e+37
      hs=dh*dh
      xs=dx*dx
      if(hs.lt.xs) return
      dy=sqrt(hs-xs)
      acosd2=atan2(dy,dx)*57.29578
      return
      end
c*****************************************************************************
      logical function inside(n,x,z,xp,zp)
c   is xp,zp inside polygon defined by x,z ?
      dimension x(1),z(1)
      logical z1,z2
      inside=.false.
      do 5 i=1,n
      i1=i+1
      if(i1.gt.n) i1=1
      dz=z(i1)-z(i)
      if(dz.eq.0.0) go to 5
      z1 = zp.le.z(i)
      z2 = zp.gt.z(i1)
      if((z1.and.z2) .or. (.not.z1 .and. .not.z2)) go to 3
      go to 5
c   points on boundary are outside with condition .lt.
c   for inclusion change to .le.
3     rslope=(x(i1)-x(i))/dz
      d=(xp-x(i))-(zp-z(i))*rslope
      if(abs(d).lt.1.e-4) d=0.0
      if(d.lt.0.0) inside = .not.inside
5     continue
      return
      end
c*****************************************************************************
      subroutine ammi(n,a,amn,amx,mn,mx,isave)
      dimension a(n)
      if(isave.gt.1) go to 1
      mn=1
      mx=1
      amn=a(1)
      amx=a(1)
1     do 3 i=1,n
      if(a(i).ge.amn) go to 2
      amn=a(i)
      mn=i
```

```
2     if(a(i).le.amx) go to 3
      amx=a(i)
      mx=i
3     continue
      return
      end
c*********************************************************************
      logical function member(itest,m,n)
      dimension m(n)
      member=.true.
      do i=1,n
      if(m(i).eq.itest) return
      enddo
      member=.false.
      return
      end
c*********************************************************************
      subroutine shsort(n,w,m,iasend)
c  shell diminishing increment sort
c  iasend < 0 for descending order
      dimension w(n),m(n)
      do 1 i=1,n
1     m(i)=i
      is=n
10    is=is/2
      if(is) 70,70,20
20    k=n-is
      i2=1
30    i=i2
40    j=i+is
      if(iasend) 42,43,43
42    if(w(m(i))-w(m(j))) 60,50,50
43    if(w(m(j))-w(m(i))) 60,50,50
50    i2=i2+1
      if(i2-k) 30,30,10
60    it=m(i)
      m(i)=m(j)
      m(j)=it
      i=i-is
      if(i) 50,50,40
70    return
      end
c*********************************************************************
      subroutine cvc(a,i2)
c  i2=1 for lower case to upper
      dimension ilb(2),iub(2),icv(2)
      character a*(*)
      data ilb/97,65/, iub/122,90/, icv/-32,+32/
      n=len(a)
      if(i2.eq.1 .or. i2.eq.2) then
      do 10 i=1,n
        j=ichar(a(i:i))
        if(j.lt.ilb(i2) .or. j.gt.iub(i2)) go to 10
        j=j+icv(i2)
```

```
        a(i:i)=char(j)
10      continue
        else
        type *,' cvc: invalid convert parameter'
        endif
        return
        end
c*************************************************************************
        function leftj(a)
c  left justifies a string and returns the position
c  of the last nonblank character
        character a*(*)
        n=len(a)
        if(a(1:1).ne.' ') go to 15
        do 1 m=2,n
1       if(a(m:m).ne.' ') go to 5
        leftj=0
        return
c
5       i2=0
        do 10 i=m,n
        i2=i2+1
        a(i2:i2)=a(i:i)
10      continue
        do 11 i3=i2+1,n
11      a(i3:i3)=' '
        n=n-m+1
c
15      do 20 leftj=n,1,-1
20      if(a(leftj:leftj).ne.' ') go to 25
25      return
        end
c*************************************************************************
        function noyes(idum)
        character inp*20
        ic=0
        noyes=-1
1       continue
        read(5,2,err=5) inp
2       format(a20)
        if(ic.ge.3) go to 9
4       if(inp(1:1).eq.'y' .or. inp(1:1).eq.'Y') noyes=1
        if(inp(1:1).eq.'n' .or. inp(1:1).eq.'N') noyes=0
        if(noyes.gt.-1) return
5       type  6
6       format(' y or n:'$)
        ic=ic+1
        go to 1
9       type *,' count exceeded, answering no'
        noyes=0
        return
        end
c*************************************************************************
        subroutine clrscr(iop)
```

```
c   iop=1 (or ne 2) exit in Tektronix mode,
c   iop=2 exit in ANSI character mode
c   "0g is a Digital Engineering Retrographics command
      call sendesc('"0g')
      call sendesc('[2J')
      call sendesc('[1;1H')
      call sendesc(char(29))
      call sendesc(char(12))
      if(iop.eq.2) call sendesc('"0g')
      return
      end
c**********************************************************************
      subroutine sendesc(tail)
      character fmt*14,esc*1,tail*(*)
      n=len(tail)
      fmt=' '
      assign 1 to label
      if(n.gt.9) assign 2 to label
      if(n.gt.99) assign 3 to label
      write(fmt,label) n
1     format( '(1x,a1,a', i1, ',$)' )
2     format( '(1x,a1,a', i2, ',$)' )
3     format( '(1x,a1,a', i3, ',$)' )
      esc=char(27)
      write(6,fmt) esc,tail
      return
      end
c**********************************************************************
      subroutine symbol(x,y,nxy,ich,size,angle)
c  symbol driver where x,y are in data units (clipping occurs outside
c  the data window).
c  'ich' is an integer akin to fortran 77 coding ich=038=ichar('&').
c  'vchar'  uses 1-13, 24-31 for symbols such as diamonds and squares,
c  and 32-126 as the normal ascii character sequence when icode=1.
      dimension x(nxy),y(nxy)
      data icode/1/, xoff,yoff/0.0,0.0/
      radian=1.7453292e-2*angle
      call vchar(x,y,ich,nxy,icode,size,radian,xoff,yoff)
      return
      end
c**********************************************************************
      subroutine text(x,y,string,size,angle)
c  text driver where x,y are in plot units.
      dimension is(25)
      character s*100
      character string*(*)
      equivalence (is,s)
      data icode/3/, xoff,yoff/0.0,0.0/
      lenstr=len(string)
      do nch=lenstr,1,-1
        ich=ichar(string(nch:nch))
        if(ich.gt.32 .and. ich.lt.127) go to 10
      enddo
      return
```

```
10      s(1:nch)=string(1:nch)
        radian=1.7453292e-2*angle
        call vchar(x,y,is,nch,icode,size,radian,xoff,yoff)
        return
        end
```